Theses and Dissertations--Electrical and Computer Engineering

2013

FPGA TO POWER SYSTEM THEORIZATION FOR A FAULT LOCATION AND SPECIFICATION ALGORITHM

Christina Yeoman
University of Kentucky, christina.yeoman@gmail.com

Recommended Citation
Yeoman, Christina, "FPGA TO POWER SYSTEM THEORIZATION FOR A FAULT LOCATION AND SPECIFICATION ALGORITHM" (2013). Theses and Dissertations--Electrical and Computer Engineering. 21.
https://uknowledge.uky.edu/ece_etds/21

This Master's Thesis is brought to you for free and open access by the Electrical and Computer Engineering at UKnowledge. It has been accepted for inclusion in Theses and Dissertations--Electrical and Computer Engineering by an authorized administrator of UKnowledge. For more information, please contact UKnowledge@lsv.uky.edu.
STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained and attached hereto needed written permission statements(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine).

I hereby grant to The University of Kentucky and its agents the non-exclusive license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless a preapproved embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student’s advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student’s dissertation including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Christina Yeoman, Student
Dr. Yuan Liao, Major Professor
Dr. Zhi Chen, Director of Graduate Studies
FPGA TO POWER SYSTEM THEORIZATION FOR A FAULT LOCATION AND
SPECIFICATION ALGORITHM

THESIS

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in the
Electrical Engineering in the College of Engineering
at the University of Kentucky

By
Christina Marie Yeoman

Director: Dr. Yuan Liao, Associate Professor of Electrical & Computer Engineering
Lexington, Kentucky
2013

Copyright© Christina Marie Yeoman 2013
ABSTRACT OF THESIS

FPGA TO POWER SYSTEM THEORIZATION
FOR A FAULT LOCATION AND SPECIFICATION ALGORITHM

Fault detection and location algorithms have allowed for the power industry to alter the power grid from the traditional model to becoming a smart grid. This thesis implements an already established algorithm for detecting faults, as well as an impedance-based algorithm for detecting where on the line the fault has occurred and develops a smart algorithm for future HDL conversion using Simulink. Using the algorithms, the ways in which this implementation can be used to create a smarter grid are the fundamental basis for this research. Simulink was used to create a two-bus power system, create environment variables, and then Matlab was used to program the algorithm such that it could be FPGA-implementable, where the ways in which one can retrieve the data from a power line has been theorized. This novel approach to creating a smarter grid was theorized and created such that real-world applications may be further implemented in the future.

KEYWORDS: Fault detection, Fault classification, Fault location, FPGA, Smart Grid.

Christina Marie Yeoman

(04/17/2013)
FPGA TO POWER SYSTEM THEORYIZATION FOR A FAULT LOCATION AND SPECIFICATION ALGORITHM

By

Christina Marie Yeoman

_____________________________________

Director of Thesis:
Dr. Yuan Liao

_____________________________________

Director of Graduate Studies:
Dr. Zhi Chen

4/17/2013
Dedications

To my parents, Dorothy Yeoman and Michael Yeoman, who have always believed in my dreams, pushed me to succeed, and who have instilled values and dedication to achieving any goal to which I may set my mind.

To my friends, who have given me the greatest happiness for their presence and light in my life.

To the Electrical and Computer Engineering professors, without whom I would not have received the most excellent education and valuable life-lessons from their advisory guidance.

To anyone along the way who has helped me, guided me, and encouraged me to succeed.

-Christina Marie Yeoman
ACKNOWLEDGEMENTS

Firstly, I would like to express how grateful I am to have had Dr. Yuan Liao as my thesis chair and advisor. I greatly admire his encouragement, patient assistance, empathy for my academic and financial situations, and skilled knowledge in my thesis topic area, without which this thesis would have been impossible.

Secondly, I would also like to thank Dr. Lawrence Holloway and Dr. Aaron Cramer for their willingness to be a part of my thesis committee and for their helpful suggestions, which allowed my thesis to become the best it can possibly be.

Thirdly, I would like to thank my parents, friends, and everyone who has encouraged and motivated me along the way.

Fourthly, I would like to thank the University of Kentucky’s College of Engineering, Electrical and Computer Engineering Department, and Student Financial Aid office that have greatly assisted and been incredibly understanding of my financial situation, without which my degree would not have been possible.

Finally, I would like to acknowledge the Power and Energy Institute of Kentucky for its financial assistance, and more importantly for giving the invaluable knowledge that I have gained by participating in this program. Without any of these invaluable components to my achieving this thesis, my degree would not have been possible.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ........................................................................................................ iii

LIST OF TABLES .................................................................................................................... vi

LIST OF FIGURES ................................................................................................................ vii

CHAPTER ONE: INTRODUCTION .................................................................................. 1
   1.1 INTRODUCTION ......................................................................................................... 1
   1.2 MOTIVATION .......................................................................................................... 4
   1.3 OUTLINE OF THE THESIS ....................................................................................... 5

CHAPTER TWO: LITERATURE REVIEW ....................................................................... 7
   2.1 FAULT TYPE CLASSIFICATION AND DETECTION TECHNIQUES ...................... 7
      2.1.1 SENSOR BASED METHODS ............................................................................. 7
      2.1.2 WAVELET BASED METHODS ......................................................................... 9
   2.2 TRANSMISSION LINE FAULT LOCATION TECHNIQUES .................................... 10
      2.2.1 TRAVELING WAVE METHODS ..................................................................... 10
      2.2.2 IMPEDANCE BASED METHODS .................................................................... 11
         2.2.2.1 SIMPLE REACTANCE METHOD ................................................................. 12
         2.2.2.2 TAKAGI METHOD ..................................................................................... 12
         2.2.2.3 MODIFIED TAKAGI METHOD .................................................................. 13

CHAPTER THREE: FPGA TECHNOLOGIES AND THEIR THEORETICAL
IMPLEMENTATION IN POWER SYSTEMS ..................................................................... 14
   3.1 INTRODUCTION TO FPGA TECHNOLOGIES ....................................................... 14
   3.2 MODEL OF AN FPGA .............................................................................................. 15
   3.3 IMPLEMENTING THE THEORETICAL ALGORITHMS .......................................... 15
   3.5 GATHERING DATA .................................................................................................. 17

CHAPTER FOUR: FAULT LOCATION AND DETECTION IN POWER
TRANSMISSION LINES USING FPGA TECHNOLOGIES ........................................... 18
   4.1 INTRODUCTION ....................................................................................................... 18
   4.2 IMPLEMENTING THE POWER SYSTEM MODEL ................................................. 18
   4.3 OUTLINE OF THE ALGORITHMS IMPLEMENTED ............................................. 21
      4.3.1 FAULT DETECTION AND CLASSIFICATION ALGORITHM AND
         IMPLEMENTATION ............................................................................................... 22
         4.3.1.1 DEFINING THE FAULT CLASSIFICATION VARIABLES ......................... 22
         4.3.1.2 DEFINING THE FAULT CLASSIFICATION ALGORITHM ......................... 24
         4.3.1.3 WRITING THE FAULT CLASSIFICATION CODE ....................................... 27
      4.3.2 FAULT LOCATION ALGORITHM AND IMPLEMENTATION ....................... 31
         4.3.2.1 DEFINING THE FAULT LOCATION VARIABLES ...................................... 31
         4.3.2.2 DEFINING THE FAULT LOCATION ALGORITHM ....................................... 34
         4.3.2.3 IMPLEMENTING THE FAULT LOCATION ALGORITHM CODE .................. 35
4.3.3 METHODOLOGY FOR CONTINUOUS FAULT DETECTION AND LOCATION 37
  4.3.3.1 INTRODUCING THE CONTINUOUS ALGORITHM VARIABLES .............................. 37
  4.3.3.2 DEFINING THE CONTINUOUS ALGORITHM ......................................................... 38
  4.3.3.3 IMPLEMENTING THE CONTINUOUS ALGORITHM CODE ............................... 39
4.4 ADJUSTING THE CODE FOR THE HDL CONVERTER ................................................. 41
4.5 IMPLEMENTING THE HDL CONVERTER .................................................................... 45
  4.5.1 RUNNING THE HDL WORKFLOW ADVISOR ....................................................... 45
4.6 OVERVIEW OF TESTING PROCESS .............................................................................. 49

CHAPTER FIVE: EXPERIMENTAL RESULTS ............................................................. 50
  5.1 INTRODUCTION TO TESTING PROCESS ................................................................. 50
  5.2 FAULT TYPE DETECTION, CLASSIFICATION, AND LOCATION TESTING STRATEGY ................................................................. 50
    5.2.1 CHANGING THE FAULT OCCURRENCE .......................................................... 51
    5.2.2 CHANGING THE FAULT TYPE ......................................................................... 52
    5.2.3 CHANGING THE RESISTANCE VALUES ........................................................... 53
  5.3 TESTING THE RESEARCH-BASED AND FAULT DETECTION AND CLASSIFICATION ALGORITHMS .......................................................................................... 54
    5.4.1 FAULT LOCATION SIMULATION 1 VERSUS SIMULATION 2 RESULTS ........ 74

CHAPTER SIX: CONCLUSIONS AND SCOPE FOR FUTURE WORK ..................... 82

Appendix A .................................................................................................................... 84
Appendix B .................................................................................................................... 97
Appendix C ................................................................................................................... 104
BIBLIOGRAPHY ......................................................................................................... 108
VITA ............................................................................................................................. 112
LIST OF TABLES

TABLE 4.1 DEFINING CONDITIONS FOR FAULT TYPE IF LINES HAVE NOT EXPERIENCED A SIGNIFICANT CHANGE............................................................................................................................................. 26
TABLE 4.2 DEFINING CONDITIONS FOR FAULT TYPE IF LINES HAVE EXPERIENCED A SIGNIFICANT CHANGE.................................................................................................................................. 26
TABLE 5.1 PHASE TO GROUND SIMULATION 1 .............................................................................. 75
TABLE 5.2 PHASE TO GROUND SIMULATION 2 .............................................................................. 75
TABLE 5.3 PHASE TO GROUND SIMULATION 3 .............................................................................. 76
TABLE 5.4 DOUBLE PHASE TO GROUND SIMULATION 1 ................................................................ 77
TABLE 5.5 DOUBLE PHASE TO GROUND SIMULATION 2................................................................... 77
TABLE 5.6 DOUBLE PHASE TO GROUND SIMULATION 3 ................................................................. 78
TABLE 5.7 PHASE-TO-PHASE SIMULATION 1 ................................................................................. 79
TABLE 5.8 PHASE-TO-PHASE SIMULATION 2 ................................................................................. 79
TABLE 5.9 PHASE TO PHASE SIMULATION 3 .................................................................................. 80
TABLE 5.10 THREE PHASE SIMULATION 1 ..................................................................................... 81
TABLE 5.11 THREE PHASE SIMULATION 2 ..................................................................................... 81
TABLE 5.12 THREE PHASE SIMULATION 3 ..................................................................................... 81
LIST OF FIGURES

FIGURE 2.1 FOC-BASED IMPLEMENTATION OF FAULT DETECTION ................................................. 9
FIGURE 3.1 A SPARTAN 6 FPGA .................................................................................................... 15
FIGURE 4.1 SCHEMATIC OF THE FAULT SIMULATION MODEL IN SIMPOWERSYSTEMS .......... 19
FIGURE 4.2 SCHEMATIC OF THE FAULTDETECTORFINAL Code .................................................... 28
FIGURE 4.3 SCHEMATIC OF THE PERFORMFFTFinal Code .......................................................... 29
FIGURE 4.4 SCHEMATIC OF THE FAULTDETECTOR1 Code ............................................................. 30
FIGURE 4.5 SCHEMATIC OF THE FAULTDISTANCE Code ............................................................... 36
FIGURE 4.6 SCHEMATIC OF THE runFAULTDETECTOR Code ....................................................... 40
FIGURE 4.7 SCHEMATIC OF THE Top LEVEL DESIGN FOR THE MODEL Code ......................... 43
FIGURE 4.8 SIMULINK VERSION OF THE RESEARCH-BASED Algorithm .................................. 44
FIGURE 4.9 SNAPSHOT OF SELECTING THE SYSTEM TO CONVERT ........................................ 46
FIGURE 4.10 SNAPSHOT OF SETTING THE FPGA PARAMETERS .................................................. 47
FIGURE 4.11 SNAPSHOT CONFIRMING THAT THE SYSTEM IS CONVERTIBLE .............................. 47
FIGURE 4.12 SNAPSHOT OF HDL CONVERSION ERRORS ............................................................. 48
FIGURE 5.1 SCHEMATIC FOR THE Top LEVEL DESIGN FOR THE SIMULATION Code ................. 56
FIGURE 5.2 SCHEMATIC FOR THE SIMULATION Code FOR runFAULTDETECTOR ...................... 57
FIGURE 5.3 Fault Type AG AT 2/60 ............................................................................................ 58
FIGURE 5.4 Fault Type AG AT 6/60 ............................................................................................. 58
FIGURE 5.5 Fault Type AG AT 10/60 .......................................................................................... 59
FIGURE 5.6 Fault Type BG AT 2/60 ............................................................................................. 59
FIGURE 5.7 Fault Type BG AT 6/60 ............................................................................................ 60
FIGURE 5.8 Fault Type BG AT 10/60 .......................................................................................... 60
FIGURE 5.9 Fault Type CG AT 2/60 ............................................................................................. 61
FIGURE 5.10 Fault Type CG AT 6/60 .......................................................................................... 61
FIGURE 5.11 Fault Type CG AT 10/60 ........................................................................................ 62
FIGURE 5.12 Fault Type ABG AT 2/60 ......................................................................................... 62
FIGURE 5.13 Fault Type ABG AT 6/60 ......................................................................................... 63
FIGURE 5.14 Fault Type ABG AT 10/60 ...................................................................................... 63
FIGURE 5.15 Fault Type BCG AT 2/60 ......................................................................................... 64
FIGURE 5.16 Fault Type BCG AT 6/60 ......................................................................................... 64
FIGURE 5.17 Fault Type BCG AT 10/60 ...................................................................................... 65
FIGURE 5.18 Fault Type CAG AT 2/60 ......................................................................................... 65
FIGURE 5.19 Fault Type CAG AT 6/60 ......................................................................................... 66
FIGURE 5.20 Fault Type CAG AT 10/60 ...................................................................................... 66
FIGURE 5.21 Fault Type AB AT 2/60 ........................................................................................... 67
FIGURE 5.22 Fault Type AB AT 6/60 ........................................................................................... 67
FIGURE 5.23 Fault Type AB AT 10/60 ........................................................................................ 68
FIGURE 5.24 Fault Type BC AT 2/60 ........................................................................................... 68
FIGURE 5.25 Fault Type BC AT 6/60 ........................................................................................... 69
FIGURE 5.26 Fault Type BC AT 10/60 ........................................................................................ 69
FIGURE 5.27 Fault Type CA AT 2/60 ........................................................................................... 70
FIGURE 5.28 Fault Type CA AT 6/60 ........................................................................................... 70
FIGURE 5.29 FAULT TYPE CA AT 10/60.......................................................................................... 71
FIGURE 5.30 FAULT TYPE ABC AT 2/60...................................................................................... 71
FIGURE 5.31 FAULT TYPE ABC AT 6/60...................................................................................... 72
FIGURE 5.32 FAULT TYPE CG AT 10/60...................................................................................... 72
CHAPTER ONE: INTRODUCTION

1.1 INTRODUCTION

Due to the overwhelming demand for energy change and improvement both environmentally and politically, the demand to make the electric power grid smart has also become a prominent theme in the advancement of technology as a whole. In short, the research being conducted both in the recent past and in the imminent future resides in updating, regulating, and inventing technologies such that our power grid may sustain itself. One of the worst hindrances to an electric power grid is the interruption and disruption of a power, thus resulting in a fault in the power system [1]. Since the demand for high efficiency, low cost, and high availability of energy on the grid are becoming extremely important, the necessity to accurately and quickly detect, classify, and locate a power grid failure, or power system fault, is also crucial [2].

The ways in which a fault might occur range from natural weather changes such as thunderstorms, lightning, hurricanes, or other natural disasters, to a momentary animal or tree contact on the line [3]. Faults occur in the transient, persistent, symmetric, or asymmetric kind. The methods in order to detect these different types of faults are unique in nature and thus there is no overarching fault detection technique. Indeed, the High Voltage Transmission Lines are more susceptible to have a fault occur than the local distribution lines. The lack of insulation on the High Voltage Transmission Lines allows for high susceptibility to a fault occurring due to the reasons listed above since these transmit power from a generating plant to a high voltage substation, where the local distribution lines are able to have insulation due to the low voltage and short line length.
Due to the transmission lines being miles long, having to run through various terrain, and having to endure such harsh weather conditions without being insulated, the length of time that it may take to check for where and what type of fault occurred, can take several minutes to several hours [4]. The type of fault that can occur could be a line to line, single line to ground, double line to ground, three phase, or three phase to ground. There have been several proposed methods for fault detection and classification [5-8]:

• High-frequency components

• Voltage and current magnitudes compared for threshold values

• Wavelet analysis

• Sensor placement for artificial detection algorithms

Indeed, there have also been some well-established methods for determining where on the line a fault may have occurred. These methods also implement similar mechanisms as fault detection and classification [9]:

• High-frequency fault based current and voltage

• Impedance measuring

• Traveling-wave phenomenon

• Artificial Intelligence

Conjunctly, there have been some efforts in the location, identification, and classification methods that have guided many efforts in creating an electric power grid to becoming smart and highly efficient [10]:

2
Though, while all of these areas have been researched extensively and made an important mark on allowing the power grid to move forward, a smart, fundamental step that has not been researched or addressed is an FPGA implementation to interact with a power system. There are a few methods that people have attempted to implement in a similar fashion [11-13]:

• Voltage Sensor to Analog to Digital Converter to FPGA program
• Power Line Phase Measurement using FPGAs
• FPGA and High-Speed Power Line Fault Classification

Finally, the basis of the research was conducted to transform a classification and detection algorithm in conjunction with a fault location method to then be able to provide a theoretical implementation and novel algorithm for combining the two algorithms such that this algorithm contains potential for a real-world application. The algorithms used to implement this design were the wavelet analysis, impedance based location measuring, and FPGA and High-Speed Power Line to Voltage Sensor to Analog to Digital Converter to FPGA Program theorization.
1.2 MOTIVATION

The importance of conducting research for a thesis relies on demonstrating a fresh outlook to already proved theories and work, thus the importance of my research is to provide a way for an already-established algorithm to detect a fault location occurrence, develop my own way of detecting a potential fault, and then implement an impedance-based technique for discovering where the fault has occurred on the power line. In doing so, and programming these ideas into a Matlab script which can then be converted into HDL for potential FPGA implementation on a power line, there provides a theoretical and novel approach to achieving crucial, needed, and necessary elements accurately.

The initial motivation of the thesis was to be able to accurately detect the fault type on a power line using a modified algorithm for both voltage and current where the power system would be simulated by a two-bus Matlab Simulink model. However, after perfecting this algorithm, it seemed more reasonable to achieve this novel method one step further, and apply fault location algorithm to it. After this step was written, the final motivation was to allow these two algorithms to be theorized into a potential real-world model. Thus, the thesis attempts to save the power companies money and the customer’s time and frustration from losing power by providing a novel algorithm which readily executes fault detection and location algorithms.

Though several other methods of how to obtain a fault location, detection, and classification have been mentioned throughout this thesis, the need to distinguish the reasoning for why I chose the methods that I did must be introduced along with the other novel approaches that have been realized in other works by fellow researchers in this
field. Thus, the importance of the selective nature of my research will demonstrate itself, and hopefully sustain itself as a way to improve the power grid, and further the “smart grid” push toward which the power engineers are working.

1.3 OUTLINE OF THE THESIS

The second chapter introduces and briefly discusses the related work that has been examined to ensure that the research conducted for this thesis was relevant. It is crucial to address all previous and current research methods in order for the research conducted for this thesis to present itself as a novel approach to improving the electric power grid.

The third chapter expands on the motivations for this thesis with a theoretical implementation of how the algorithms developed for this thesis could be implemented with a High Voltage Transmission line.

The fourth chapter identifies the methodologies and explains the ways in which they were implemented, from a conceptual idea to a fully realized Matlab coding implementation. The coding methods are explained, as well as the introduction of how the algorithms become realized in the Matlab to HDL theorization.

In the fifth chapter, the testing of both algorithms is conducted by running the code simulation algorithm, such as that the fault occurrence time, the ground resistance, line resistance, fault type, and fault location would be altered. In conducting these tests, the algorithms in the code demonstrate that they are ready for FPGA implementation.
In the seventh chapter, the conclusion, research is summarized and the work that has been conducted for this thesis is examined. Future work for this thesis is discussed.
CHAPTER TWO: LITERATURE REVIEW

In order to embark upon research, it is first relevant and key to introduce and understand the similar work that is being done in the area that you are researching. In this chapter, the relative work emphasis touches briefly on the other types of algorithms that may be implementable, including a mention the type that this thesis implemented. In doing this, then any future work that could be conducted would be to test the different algorithms on the simulations so that the best potential HDL conversion and implementation occurs.

2.1 FAULT TYPE CLASSIFICATION AND DETECTION TECHNIQUES

There have been several approaches developed to detecting and classifying faults on a power transmission line. The ones considered before moving forward with the research were based on using sensors and wavelet transforms, where sensors and wavelets were a part of the detection algorithm, rather than used for an algorithm. The algorithm implemented for this thesis uses a wavelet approach for classification and detection.

2.1.1 SENSOR BASED METHODS

There are several different methods by which one can implement a sensor to detect the fault type, such as implementing fiber optics [14], implementation for artificial neural network detection [16], and using fuzzy techniques for sensor selection [15].
However, the fiber-topics approach would be a great alternative to the fault detection wavelet algorithm implemented in this paper, so it is further discussed below.

Fiber-optics Current Sensors (FOCS) essentially transform the magnetic field of the current on the line into a mechanical strain on a fiber Bragg grating (FBG). In this way, these sensors can allow for multiplexing of the fault signals, and thus replaces the conventional Current Transformer (CT) [17-19].

The magnitude of the frequency response of the magnetic field sensitivity (H) can be found with the following equation, where \( \lambda_1 = 1.560475 \mu m \) and \( \lambda_2 = 1.5576 \mu m \) are the lower and upper limits of the spectrum.

\[
R_{avg} = \frac{1}{\lambda_2 - \lambda_1} \int_{\lambda_1}^{\lambda_2} R_T(\lambda)d\lambda
\]  

With this in mind, the FBGs can be located at a substation and operate independently, then conjunctively to allow for signal messaging based on the frequencies that are emitted, and detected running down the line, where the fault detection would then use a flagging mechanism to indicate that a fault just occurred. The FBG’s line width and its product control the FOCS output. Fig 2.1 demonstrates a 16-type fault detection FOCS on a 20 mW broadband light source when a superluminescent light emitting diode scans the whole network.
2.1.2 WAVELET BASED METHODS

Using a wavelet-based method is a common approach to fault detection, where the analysis of the three-phase current and voltage data such as using a moving window of the signal are used to determine if there is a fault on the line [20]. Other techniques include using a filter bank for creating indicators [21], as well as using superimposed ratios to find the fault type [7]. Since [7] has already been implemented in the fault detection algorithm for this research, [20] will be discussed as an alternative approach to wavelet based methods for fault detection.

The basis of the method introduced in [20] is to use a continuous wavelet transform, such that it is defined in (2).

\[
L \psi f(a, t) = \frac{1}{\sqrt{\pi} a} \int_{-\infty}^{\infty} \psi \left( \frac{u-t}{a} \right) f(u) du (a \neq 0), t \in R
\]  

(2)
Where a finite energy function must fulfill the admissibility condition.

\[ C_\psi := 2\pi \int_{-\infty}^{+\infty} \frac{|\Psi(\omega)|}{|\omega|} d\omega < \infty \]  

(3)

2.2 TRANSMISSION LINE FAULT LOCATION TECHNIQUES

There have been several proposed methods in this area that would help the power companies understand approximately where the fault transmission line occurred. There is a huge push for obtaining the best algorithm possible to assess the fault location with high accuracy. Several authors have made an effort in this area [22-29]. The fundamental techniques that have been heavily studied are: traveling wave, neural networks, and impedance-based methods.

2.2.1 TRAVELING WAVE METHODS

The idea behind a traveling-wave based method is to examine the fault locator by the high-frequency traveling wave [9]. The goal of this section is to give a basic overview of how it works. Essentially, the way it works is by correlating the forward and backward traveling waves on the transmission line, as several have proceeded to do in their work [30-32].

If one considers ‘l’ to be length, ‘v’ to be velocity, \( L' \) to be the inductance per unit length, and \( C' \) to be the characteristic impedance of \( Z_c \), then the fault at distance ‘x’ of terminal A, where A is the positive end of the spectrum of a single phase lossless
transmission line, for example, then the voltage and current values can be found in (4) and (5).

\[
\frac{\partial e}{\partial x} = -L' \frac{\partial i}{\partial t} \tag{4}
\]

\[
\frac{\partial i}{\partial x} = -C' \frac{\partial e}{\partial t} \tag{5}
\]

Such that the solution yield (6) and (7).

\[
e(x,t) = e_f(x-\nu t) + e_r(x+\nu t) \tag{6}
\]

\[
i(x,t) = \frac{1}{Z_c} e_f(x-\nu t) - \frac{1}{Z_c} e_r(x+\nu t) \tag{7}
\]

In order to determine the time it takes to travel from the fault to the discontinuity \(\tau_A\) and \(\tau_B\), one can use a GPS device. If \(c\) is the wave propagation constant, 299.79 m/sec, then the fault location can be found by (8).

\[
x = \frac{l-c(\tau_A-\tau_B)}{2} \tag{8}
\]

2.2.2 IMPEDANCE BASED METHODS

Impedance based methods rely on the fault resistances not being very high, otherwise they will not detect the fault location accurately [33]. Depending on the number of terminals that the voltage and current are being collected, there are two different methods by which the data can be collected. The idea is to look at the impedance values from one (or multiple) end and calculate them from there.
2.2.2.1 SIMPLE REACTANCE METHOD

This is a slightly modified version of the algorithm, which has been implemented in the research for this thesis.

The basic idea of a model, where A is on the left end and B is on the right end of the terminals, and impedance values for both terminals are set, then the voltage and current measurement would occur after the impedance has been established. The measurement values for voltage and current have been declared and that the impedance has been found such that \( V_a \) is the voltage at terminal A, \( x \) is the distance to the fault from A, \( I_a \) is the current flowing out of terminal A, \( V_f \) is the fault voltage, and \( Z_l \) is the line impedance that has been calculated. The following equations (11) and (12) are the steps commonly implemented after the calculation of the line impedance has occurred [34].

\[
V_A = x \cdot Z_L \cdot I_A + V_f \quad (11)
\]

\[
V_A = x \cdot Z_L \cdot I_A + R_f \cdot I_f \quad (12)
\]

Such that the fault current \( I_f \) and the fault resistance \( R_f \) can be used to find the fault location.

2.2.2.2 TAKAGI METHOD

The Takagi method [35] is a commonly used method which requires both the pre-fault and fault data to find the fault location, but because of this, it simplifies the reliability on low fault resistance values. Thus, the fault resistance equation can be give by (13).

The Fault Resistance is given by

\[
R_f = \frac{V_A - Z_C I_A \tanh y x}{(\frac{V_A}{Z_C} \tanh y x - I_A') \psi e^{j \theta}} \quad (13)
\]
Where the parameters for this equation are the same as in 2.2.2.1: \( I_f \) is the fault current, \( R_f \) if the fault resistance, \( V_a \) is the voltage at terminal A, \( x \) is the distance to the fault from A, \( I_a \) is the current flowing out of terminal A, \( V_f \) is the fault voltage, and \( Z_l \) is the line impedance that has been calculated. Thus, \( x \) can be found in (14) when (15).

\[
Z_L = \gamma Z_C
\]  

(14)

And \( x = \frac{Im(V_A, I_A^*)}{Im(Z_L I_A, I_A^*)} \) is the distance to the fault from terminal A.  

(15)

2.2.2.3 MODIFIED TAKAGI METHOD

Instead of the Tagaki method’s reliance on pre-fault and after-fault data, the modified method does not require any pre-fault data since it uses the zero-sequence current instead of super position for ground faults [36].

Given that \( I_R \) is the zero-sequence current, \( \beta \) is the zero-sequence current angle, \( Z_{1L} \) is the positive sequence line impedance, \( V_A \) is voltage measured at terminal A, \( I_A \) is the flowing out of terminal A, and the location of the fault can be represented by ‘\( x \)’ in equation (16).

\[
x = \frac{Im(V_A I_R^* e^{-j\beta})}{Im(Z_{1L} I_A I_R^* e^{-j\beta})}
\]  

(16)
CHAPTER THREE: FPGA TECHNOLOGIES AND THEIR THEORETICAL IMPLEMENTATION IN POWER SYSTEMS

3.1 INTRODUCTION TO FPGA TECHNOLOGIES

A Field-Gate Programmable Array is a board that consists of electrical components such as transistors, resistors, capacitors, and etcetera. However, the most fundamental components of these boards are the processors that sit on them. This makes the FPGAs capable of being programmed with a list of instructions so that the processor instructs all of the sub-components to behave a certain way. Beyond the processor, all FPGAs currently on the market have switches, a VGA port, USB ports, and even an Ethernet port. The possibilities are endless with these devices, so the research focus to create a real-world program would be excellent on an FPGA.

The purpose of their introduction here is to theorize a way in which they might be implemented on a High Voltage Transmission Line, where the line theorized in the system theorized in this thesis is a three-phase system such that one FPGA device could manage a particular element of such a power line, using the inputs and a few extra tools to assist in allowing this research to become realized.
3.2 MODEL OF AN FPGA

An FPGA chip can be seen in Image 3.1, where the board and its several components are present [37]. This FPGA is in the Spartan 6 Family.

![Spartan 6 FPGA](image)

**Figure 3.1 A Spartan 6 FPGA**

3.3 IMPLEMENTING THE THEORETICAL ALGORITHMS

In future sections, the algorithms developed to interface with the FPGA will be introduced. Every FPGA has Input/Output ports such that it can read and write data out. In the case of the focus of this research and all power system applications, the FPGA will need to be reading voltage, current, resistance, and outputting a desired output. Most ports would not need to be used, unless a switch was needed to constantly enable something.
The code must be developed in either Verilog or VHDL. These programming languages are used to specifically program FPGAs, so any code that is used to potentially implement on an FPGA would need to be converted to one of the two languages. Thus, implementing the specified algorithms would take a conversion tool, or the programmer must write in Verilog themselves.

3.4 INTERFACING WITH POWER SYSTEMS

An FPGA implementation on a real-world power system would require several components in order to make such a system work. The power system along the transmission line would need to have a Current Transformer (CT) to not only read the data off of the line, but to step down the current. There has been several notable research [38-39] which has tested the effectiveness of CTs on high-power transmission lines and their results are sufficient for the type of work that this research needs to have done if it is going to be moving forward.

Additionally, CTs have also been used in a similar fashion than the goal of this thesis, where they are being implemented for fault location algorithms [40]. With these important items in mind, technology is advancing that having the power grid be processor-dependent for fault detection could be the way of the future.

Therefore, if the CT were to be implemented, then one each would be snapped to each phase such that it could step down the current immensely. The current would have to be stepped down further and then able to be fed in to the FPGA, through an Analog to Digital Converter.
The next step in building a fault detector, classifier and locator would be to buy an analog to digital converter, since the FPGA does not take in valid data points without it. Finally, establishing that the current readings feeding into the array are below the maximum possible current and voltage going in, and then the digital signal can work efficiently.

Lastly, the FPGA would need some kind of protective case so that it could avoid most, if not all, weather conditions.

3.5 GATHERING DATA

The way that an FPGA could be implemented to gather data from the power line is to first become in-sync with the power line frequency, samples per cycle, and other parameters that are going to need in order to be synced before turning it on to test if it works.

Once the FPGA has been synced with the power transmission line, then data should be read continuously, checked against the fault detection and classification parameters such that when a fault occurs, an output to a displace can say where, when, and what.
4.1 INTRODUCTION

In previous chapters, the fault detection and location have been introduced as smart ways to detect the needed fault parameters for fault detection on power transmission lines. In order to applicably implement such algorithms that have been utilized in this thesis, a Field-Programmable Based Array (FPGA) can be used to aid in transmission line fault location.

In this chapter, the algorithms which use a theoretically implementable FPGA are discussed in detail. And, to be effectively translated to HDL code, there must be a model to simulate, so the detail of how it was built is also discussed. Thus, the algorithms are introduced, the way the code was constructed is explained, and then the potential conversion to HDL is discussed.

4.2 IMPLEMENTING THE POWER SYSTEM MODEL

A MALTAB SimPowerSystems model was used to implement the power system prototype for this research, where the 500kV transmission line is 200 kilometers long. The model consists of two thirty-degrees three-phase generators that are located on each end of the line. The next fundamental components are the impedance blocks and the line length parameters. In between the line link parameter on source 1 and ZP (where the other parameter is ZQ), there is a measurement block, which is used to output the data. Finally, in between the line blocks, there is a fault simulator, where one can pick which
type of fault to simulate, along with when the fault is to occur, and the resistance values the fault should have. Figure 4.1 is the model described. The top right blocks are for breaking apart the simulation voltage and current into arrays to then be outputted to the workspace, where it outputs in p.u.

![Image of the fault simulation model in SimPowerSystems](image)

**Figure 4.1 Snapshot of the fault simulation model in SimPowerSystems.**

Introducing the parameters of the model that were used to implement the simulation are crucial to moving forward in understanding the algorithms which have been implemented in this research. Outlined below with each block heading are the necessary parameters for the rest of the algorithm implementation. If the result is represented with a ‘to’ this means that it was changed for each simulation and testing.
Simulate

Run Time (s): 10

Total Run Time (s): 60

Line 1 and Line 2

Resistance Per Unit Length (Ohms/km): [0.249168 0.60241]/1.6093

Inductance Per Unit Length (H/km): [0.00156277 0.0048303]/1.6093

To Workspace

Samples Per Cycle: 128

Frequency (Hz): 60

Base Voltage 5e5 V Three-Phase V-I Measurement

Voltage Measurement: Phase to Ground, signal label: vabc

This represents the voltage from line’s A, B, and C with an
array size of the Run (Time * Frequency) +1. The output is based
off of the nominal voltage being
500e3 (Vrms phase to phase) and Base Power being 100e6 (VA 3
phase)

Current Measurement: Yes, signal label: iabc

This represents the voltage from line’s A, B, and C with an
array size of the Run (Time * Frequency) +1. The output is based
off of the nominal voltage being
500e3 (Vrms phase to phase) and Base Power being 100e6 (VA 3
phase)
Three Phase Fault

Fault type selection (based on three phase): AG, BG, CG, AB, BC, CA, ABG, BCG, CAG, ABCG, ABC

Fault Resistances (ohms): 0.01 to 5

Ground Resistance (Rg ohms): 1 to 100

Transition times (s): [(2/60 to 10/60) 1]

Nominal voltage used for pu measurement (Vrms phase-phase): 500e3

Base Power (VA 3 phase): 100e6

The simulations were run on changing the different Three Phase Fault parameters. Fourteen simulations were run on the Fault Resistances and Ground Resistances collectively, while the transition time stayed the same (as it does not affect the fault location). And, to prove that the fault was detected based on an input/output model of graphs generated, there were thirty simulations run (at three different times for all 10 possible fault types for this model).

4.3 OUTLINE OF THE ALGORITHMS IMPLEMENTED

There were two primary algorithms that aided this research in reaching its full potential. The first is the fault detection and classification algorithm, based on superimposed voltages and currents, presented by W. Fan and Y. Liao [41]. The second is an impedance-based method, presented by J. L. Blackburn [42].
4.3.1 FAULT DETECTION AND CLASSIFICATION ALGORITHM AND IMPLEMENTATION

This algorithm uses ratios and comparators to those ratios to determine whether or not there is a fault on the line. Since it is easier to read the current from a power line than the voltage, this research went in the direction of solely using the current-based part of the algorithm. The variables presented by W. Fan and Y. were implied in their work to have already been run through a Fast-Fourier Transform algorithm, such that $I_a$, $I_b$, $I_c$, $I_{an}$, $I_{bn}$, and $I_{cn}$ were already defined [41]. The following section introduces how, in the research, the variables were defined such that the algorithm could be implemented.

4.3.1.1 DEFINING THE FAULT CLASSIFICATION VARIABLES

Below are the necessary parameters that were defined in the MATLAB script which modeled this algorithm that is later introduced. The fault occurrence, while it is defined in the simPowerSystems model, is not set in the final algorithm implementations, though for this algorithm implementation, the fault occurrence (point in time) is assumed to be something which is manually input, as the algorithm requires. The array size from the SimPowerSystems model is defined by (1)

\[
\text{Array size} = (\text{Run Time} \times \text{Samples Per Cycle}) + 1
\] (1)

In order to produce ratios, the following variables had to be created from the SimPowerSystems array ‘iabc’ and then run through the written Fast Fourier Transform function. The variables and how they are generated are listed below.
\( I_a, I_b, I_c \): phase a, b, c current phasor during the fault. In order to find the Before Fault Occurrence, the equation (2) must be executed.

\[
\text{Before Fault Occurrence} = (\text{Fault Occurrence} \times \text{Total Run Time} \times \text{Samples Per Cycle}) - \text{Samples Per Cycle}
\]  

(2)

Then, the Fast-Fourier Transform (FFT) is performed to create the magnitude and phase that were found on that entire cycle, where the starting point would be (3) and (4).

\[
\text{FFT starting point: Before Fault Occurrence + 1}
\]  

(3)

\[
\text{FFT ending point: Before Fault Occurrence + 1 + Samples Per Cycle}
\]  

(4)

The FFT function is programmed to perform the FFT on the array that is the input, and then output the fundamental phasor in the second value of the magnitude and phase arrays that are generated. Finally, the \( I_a \), \( I_b \), and \( I_c \) phasors must be found, where (5), (6) and (7) hold.

\[
I_a = (\text{second element of the magnitude of } I_a) + j \cdot e^{\text{second element of the phase angle of } I_a}).
\]  

(5)

\[
I_b = (\text{second element of the magnitude of } I_a) + j \cdot e^{\text{second element of the phase angle of } I_a}).
\]  

(6)

\[
I_c = (\text{second element of the magnitude of } I_a) + j \cdot e^{\text{second element of the phase angle of } I_a}.
\]  

(7)

\( I_0 \): zero-sequence current phasor during the fault. This was found by using the \( I_a \), \( I_b \), and \( I_c \) generated above in (8).

\[
I_0 = (I_a + I_b + I_c) / 3
\]  

(8)
\( I_{an}, I_{bn}, I_{cn} \): phase a, b, c current phasor preceding the fault. In order to find the After Fault Occurrence, equation (9) must be executed.

\[
\text{After Fault Occurrence} = (\text{Fault Occurrence} \times \text{Total Run Time} \times \frac{\text{Samples Per Cycle}}{2}) + (2 \times \text{Samples Per Cycle}) \tag{9}
\]

Then, the Fast-Fourier Transform (FFT) is performed to create the magnitude and phase were found on that entire cycle, where the starting point would be (10) and (11).

\[
\text{FFT starting point: After Fault Occurrence} + 1 \tag{10}
\]

\[
\text{FFT ending point: After Fault Occurrence} + 1 + \text{Samples Per Cycle} \tag{11}
\]

The FFT function is programmed to perform the FFT on the array that is input and have the result be stored in the second value of the magnitude and phase arrays that are generated. Finally, the \( I_{an}, I_{bn}, I_{cn} \) phasors must be found, where (12), (13), and (14) holds.

\[
I_{an} = \text{second element of the magnitude of } I_{an} + j \times e^{\text{second element of the phase angle of } I_a}. \tag{12}
\]

\[
I_{bn} = \text{second element of the magnitude of } I_{bn} + j \times e^{\text{second element of the phase angle of } I_a}. \tag{13}
\]

\[
I_{cn} = \text{second element of the magnitude of } I_{cn} + j \times e^{\text{second element of the phase angle of } I_a}. \tag{14}
\]

4.3.1.2 DEFINING THE FAULT CLASSIFICATION ALGORITHM

This section does not represent the work of the researcher, but simply redefines the algorithm which has been presented by W. Fan and Y. Liao [41].
The following equations are used for threshold values (15), (16), and (17).

\[ \varepsilon_1 = 0.3 \left| I_{an} \right| \] indicates significant change \hspace{1cm} (15)

\[ \varepsilon_2 = 1.4 \] judges the ratio between two values \hspace{1cm} (16)

\[ \varepsilon_3 = 0.1 \] used for zero-sequence comparisons \hspace{1cm} (17)

Since Ia, Ib, Ic, Ian, Ibn, and Icn have been defined, then the absolute value must be taken so that the final produce can be compared to the threshold values.

\[ D_a = \left| I_a - I_{an} \right| \] \hspace{1cm} (18)

\[ D_b = \left| I_b - I_{bn} \right| \] \hspace{1cm} (19)

\[ D_c = \left| I_c - I_{cn} \right| \] \hspace{1cm} (20)

Now, all necessary variables have been presented to be compared. The fault type classifications are as follows: AG, BG, and CG represent phase A, B, and C ground fault; AB, BC, and CA represent the ungrounded phase A to phase B fault, phase B to phase C fault, and phase C to phase A fault; ABG, BCG, and CAG represent the phase AB to ground fault, phase BC to ground fault, and phase CA to ground fault; and ABC and ABCG represent the three phase ungrounded and grounded faults.

The following tables use conditions to output the fault type, where all conditions in the conditions block must be satisfied in order to detect a fault.

If the condition \( D_a \geq \varepsilon_1, D_b \geq \varepsilon_1, D_c \geq \varepsilon_1 \) is false, then check the following conditions.
Table 4.1 Defining conditions for fault type if lines have not experienced a significant change

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Fault Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Da &gt;= ε1, Db&lt; ε1, Dc&lt;ε1</td>
<td>AG</td>
</tr>
<tr>
<td>Da &gt;= ε1, Db&gt;&gt; ε1, Dc&lt;ε1, Da/Db &gt;=ε2</td>
<td>AG</td>
</tr>
<tr>
<td>Dc &gt;= ε1, Da&gt;&gt; ε1, Db&lt;ε1, Da/Dc &gt;=ε2</td>
<td>AG</td>
</tr>
<tr>
<td>Db &gt;&gt; ε1, Da&lt; ε1, Dc&lt;ε1</td>
<td>BG</td>
</tr>
<tr>
<td>Da &gt;= ε1, Db&gt;&gt; ε1, Dc&lt;ε1, Da/Da &gt;=ε2</td>
<td>BG</td>
</tr>
<tr>
<td>Dc &gt;= ε1, Da&gt;&gt; ε1, Da&lt;ε1</td>
<td>CG</td>
</tr>
<tr>
<td>Db &gt;= ε1, Dc&gt;&gt; ε1, Da&lt;ε1, Db/Db &gt;=ε2</td>
<td>CG</td>
</tr>
<tr>
<td>Da &gt;&gt; ε1, Da&gt;&gt; ε1, Db&lt;&lt;ε1, Da/Da &gt;=ε2</td>
<td>AB</td>
</tr>
<tr>
<td>Db &gt;= ε1, Da&gt;&gt; ε1, Da&lt;ε1, Db/Db &gt;&gt;ε2, Db/Da &lt;&lt;ε2, D0&lt;&lt;ε3Da</td>
<td>AB</td>
</tr>
<tr>
<td>Da/Da &gt;&gt;ε2, Da/Dc &gt;&gt;ε2</td>
<td>AG</td>
</tr>
<tr>
<td>Db/Db &gt;&gt;ε2, Db/Dc &gt;&gt;ε2</td>
<td>BG</td>
</tr>
<tr>
<td>Dc/Da &gt;&gt;ε2, Dc/Db &gt;&gt;ε2</td>
<td>CG</td>
</tr>
<tr>
<td>Da/Dc &gt;&gt;ε2, Db/Dc &gt;&gt;ε2, D0 &gt;&gt;ε3Da</td>
<td>ABG</td>
</tr>
<tr>
<td>Da/Da &gt;=ε2, Da/Dc &gt;=ε2, D0 &gt;&gt;ε3Da</td>
<td>ABC</td>
</tr>
</tbody>
</table>

If the condition Da >= ε1, Db>= ε1, Dc>= ε1 is true, then check the following conditions.

Table 4.2 Defining conditions for fault type if lines have experienced a significant change

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Fault Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Da/Da &gt;&gt;ε2, Da/Dc &gt;&gt;ε2</td>
<td>AG</td>
</tr>
<tr>
<td>Db/Db &gt;&gt;ε2, Db/Dc &gt;&gt;ε2</td>
<td>BG</td>
</tr>
<tr>
<td>Dc/Da &gt;&gt;ε2, Dc/Db &gt;&gt;ε2</td>
<td>CG</td>
</tr>
<tr>
<td>Da/Dc &gt;&gt;ε2, Db/Dc &gt;&gt;ε2, D0 &gt;&gt;ε3Da</td>
<td>ABG</td>
</tr>
<tr>
<td>Da/Da &gt;&gt;ε2, Da/Dc &gt;&gt;ε2</td>
<td>CAG</td>
</tr>
<tr>
<td>Da/Dc &gt;&gt;ε2, Da/Da &gt;&gt;ε2</td>
<td>AB</td>
</tr>
<tr>
<td>Da/Da &gt;&gt;ε2, Da/Dc &gt;&gt;ε2, D0 &gt;&gt;ε3Da</td>
<td>ABC</td>
</tr>
<tr>
<td>Db/Db &gt;&gt;ε2, Da/Da &gt;&gt;ε2, D0 &gt;&gt;ε3Da</td>
<td>ABCG</td>
</tr>
</tbody>
</table>
Thus, if there were a fault on the line, all fault types would be detected using this algorithm.

4.3.1.3 WRITING THE FAULT CLASSIFICATION CODE

The programs that were written to detect and classify the fault type on the line can be found in Appendix A. These are presented from the top level down in the appendix, as well as the schematics that are featured the following pages.
Function: faultDetectorFinal

Description This is a single-run function that uses the faultOcc variable to check if a fault has occurred on the power line. The fault type is outputted in an integer value so that it may be parsed later. Note that the fault occurrence must be an input so that this algorithm can run properly.

Figure 4.2 Schematic of the faultDetectorFinal Code
**Function:** performFFTFinal

**Description** This performs the Fast-Fourier Transform as described in section 4.3.1.1 of the thesis document. It is used to create the fundamental phasor’s magnitude and voltage based off of an entire cycle of data.
**Function**: faultDetectorI

**Description** Runs the algorithm specified in section 4.3.1.2, which has been developed by W. Fan and Y. Liao[40]. It uses comparators and the threshold values to yield the fault type on the line if there is one. If not, it will output a number signaling that there wasn’t a fault.

![Diagram of faultDetectorI](image)

**Figure 4.4 Schematic of the faultDetectorI Code**
In summary of this algorithm, it can be said that the SimPowerSystems simulation of the variable ‘iabc’ was generated in p.u., as we will later see in this chapter. But, this does not affect the algorithm. Since the algorithm uses ratios and threshold values to determine whether or not there is a fault on the line, then whether the current is in p.u., or in A doesn’t matter. Thus, this algorithm and the code used to generate could be used in real-world applications.

4.3.2 FAULT LOCATION ALGORITHM AND IMPLEMENTATION

The algorithm for specifying the fault location is an impedance-based algorithm. The way that it works is that it uses a ratio of voltage to positive-sequence, sequence, and current. The parameters which were necessary in order to implement this algorithm were mentioned in section 4.2, as the positive-sequence impedance and the zero-sequence impedances. In addition, the current parameters after the fault (I_a, I_b, and I_c) after performing the FFT function, whose methods were mentioned in 4.3.1, will be needed in addition to the same voltage parameters (V_a, V_b, and V_c).

4.3.2.1 DEFINING THE FAULT LOCATION VARIABLES

Though previously defined in section 4.2, the following parameters are needed. They were defined in Line 1 and Line 2 in the SimPowerSystems model.

*Resistance Per Unit Length(Ohms/km):* \([0.249168 \ 0.60241]/1.6093\)

*Inductance Per Unit Length(H/km):* \([0.00156277 \ 0.0048303]/1.6093\)
In order to turn these into positive-sequence and zero-sequence impedances, the following steps must be followed. Since the unit length is in km, then the line length must be presented in km as well. In the case of the SimPowerSystems model, the line length is **200 km**. In addition, the frequency must also be known, in this case it is **60 Hz**. Thus, the following calculations (21) and (22) can be generated, given that the Resistance Per Unit Length is defined as 1x2 array, R, and the Inductance Per Unit Length is defined as a 1x2 array, L.

\[
\text{Positive-Sequence Impedance} = \text{Line Length} \times (R[1] + (j \times 2 \pi \times \text{frequency}) \times L[1]) \tag{21}
\]

\[
\text{Zero-Sequence Impedance} = \text{Line Length} \times (R[2] + (j \times 2 \pi \times \text{frequency}) \times L[2]) \tag{22}
\]

In the future, the Positive-Sequence Impedance will be called \(Z_1\), and the Zero-Sequence Impedance will be called \(Z_0\). Since the steps to find \(I_a\), \(I_b\), and \(I_c\) have already been discussed in 4.3.1.1, then the methodology will not be re-explained here. However, one must note that \(V_a\), \(V_b\), and \(V_c\) can be found in the exact same way, and these values are equally as crucial for the algorithm. Using these variables that have been defined, then two critical variables can be defined. If you let the Zero Sequence Current Phasor During the Fault be \(I_0\), then the following condition (23) holds.

\[
I_0 = (I_a + I_b + I_c) / 3 \tag{23}
\]

And, a variable commonly called \(I_{\Sigma}\) which is used for all line-to-ground faults is listed below in (24).

\[
I_{\Sigma} = I_0 \times (Z_0 - Z_1) / Z_1 \tag{24}
\]
In addition, the fault location algorithm assumes that the fault has already been detected and the location in the array has been stored. While the location algorithm can run independently without the fault detection and classification algorithm, the two algorithms needs are similar, so it is wisest to run the fault classification algorithm is first.

Before continuing to defining the algorithm, it must be noted that in this particular SimPowerSystems model, the outputs are in p.u., thus a slight adjustment had to be made when running the code so that the positive-sequence impedance and the zero-sequence impedance have also been converted to p.u. so that everything is of the same unit. Thus, the following calculations had to be made for simulation purposes only (as this code is not included in the HDL conversation).

Two necessary components were needed from the SimPowerSystems model, in the Three Phase VI Measurement block.

*Nominal voltage used for pu measurement (Vrms phase-phase): 500e3*

*Base Power (VA 3 phase): 100e6*

After these were found, then the equation for converting the positive-sequence and zero-sequence impedance is the following (25) and (26), where the non-based impedances are the positive and zero-sequence parameters.

\[
\text{Base Impedance} = \frac{(\text{Nominal voltage})^2}{\text{Base Power}} \quad (25)
\]

\[
\text{Based Impedance} = \frac{\text{Non-based Impedances}}{\text{Base Impedance}} \quad (26)
\]
Thus, with these new base impedances, the simulation results will be properly represented.

4.3.2.3 DEFINING THE FAULT LOCATION ALGORITHM

The impedance-based algorithm implements different equations based on what grouping of faults have occurred. The groupings are: three-phase, phase-to-phase, phase-to-ground, double-phase-to-ground, and three-phase-to-ground. The algorithm presented henceforth is a modified version of J.L. Blackburn’s impedance-based algorithm [42]. For phase-to-phase faults, \( V_1 \) and \( V_2 \) are used to represent the first and then second phase. For example, if the fault is AB, then A is \( V_1 \) and B is \( V_2 \). This also follows for line-to-ground faults, where an AG fault would be represented by \( V_1 \) being A. In modeling it this way, it simplifies the algorithm.

**Three-Phase Fault**

Distance = \( \frac{V_a}{Z_1 \times V_a} \)

**Phase-to-phase and Double-Line-to-Ground Faults**

Distance = \( \frac{(V_1 - V_2)}{(Z_1 \times (I_1 - I_2))} \)

**Single-Line-to-Ground Faults**

Distance = \( \frac{V_1}{Z_1 \times (I_1 + I_2)} \)

**Three Phase to Ground Fault**

Distance = \( \frac{V_1}{Z_1} \)
This algorithm is created such that the distance it yields is in the unit that the positive-sequence and zero-sequence impedances have set in the beginning. For example: due to the SimPowerSystems simulation’s outputs of vabc and iabc being in p.u., then the $Z_0$ and $Z_1$ had to be converted to p.u. and then run through this fault detector. The output for the simulations has been left in p.u.

4.3.2.3 IMPLEMENTING THE FAULT LOCATION ALGORITHM CODE

The fault location code is to be implemented by running two programs, where one of them has already been defined in 4.3.1.3. In this research, the function “faultDetectorFinal” is used to generate the after fault parameters, $I_a$, $I_b$, $I_c$, $V_a$, $V_b$, and $V_c$. The full representation of the code can be found in Appendix A. However, the fault detection algorithm is not dependent on this function as these values can be manually inputted. These parameters are created in faultDetectorFinal as real and imaginary representations, with each phase respectively, such that it generates a 1x6 array such that the following applies.

$$I_{cycForLoc} = [\text{real}(I_a) \text{ imag}(I_a) \text{ real}(I_b) \text{ imag}(I_b) \text{ real}(I_c) \text{ imag}(I_c)];$$

$$V_{cycForLoc} = [\text{real}(V_a) \text{ imag}(V_a) \text{ real}(V_b) \text{ imag}(V_b) \text{ real}(V_c) \text{ imag}(V_c)];$$

However, in order to accurately use $I_a$, $I_b$, $I_c$, $V_a$, $V_b$, and $V_c$, a fault had to have been detected or set in the input of ‘faultDetectorFinal’ so that the right arrays were used to find the fault distance. Thus, with this in mind, the programs were created and the following schematic demonstrates the visual demonstration of them.
**Function:** faultDistance

**Description** Uses the voltage and current values of the fundamental phasor during a fault to describe where the fault has occurred on the line.

---

**FIGURE 4.5 SCHEMATIC OF THE FAULTDISTANCE CODE**
4.3.3 METHODOLOGY FOR CONTINUOUS FAULT DETECTION AND LOCATION

While the fault detection classification and location algorithms have been introduced, it would be unrealistic to input when the fault occurred to run the algorithms. So, the next step of this research was to create an algorithm such that it would detect the fault at any time, given that it knew the run time and the samples per cycle. Before going further, it is important to note that the algorithm itself was particularly developed in mind that a module or external program would be calling it continuously. Thus, it relies on the input arrays of vabc and iabc (which were created in the SimPowerSystems model), and assumes that the arrays, as input, have been filled and are ready to check for faults. Thus, this continuously runs through the array, and is set up to continuously run with new arrays as input to be checked for faults.

4.3.3.1 INTRODUCING THE CONTINUOUS ALGORITHM VARIABLES

All variables to run this algorithm have been defined in sections 4.3.1 and 4.3.2, where the following variables are needed.

- $iabc$, the current array generated directly from the SimPowerSystems model
- $vabc$, the current array generated directly from the SimPowerSystems model
- $Z0$, the calculated zero-sequence impedance
- $Z1$, the calculated zero-sequence impedance
- runtime, gathered from the SimPowerSystems model
- totalRunTime, gathered from the SimPowerSystems model
While all of these variables have been generated from the SimPowerSystems model, this exemplifies that these variables can be set and dependent on the real power system that this continuous algorithm can be run on.

4.3.3.2 DEFINING THE CONTINUOUS ALGORITHM

The algorithm iterates through the vabc and iabc variables in a way so that the fault occurrence was tested based on the run time and iterated in a for-loop. This way, the code could potentially be implemented on a continuously-running FPGA device. Since the fault classification algorithm requires that there are two whole cycles after it, and then the array can only read until three minus the set run time.

1) For every m, where m is initialized to 1 and run as long as the run time, set the temporary fault occurrence to that m.
   a. Run the fault detector code, output the fault type and the fundamental phasors for voltage and current
   b. Save each output into its respective array for each iteration

2) For every n, where n is initialized to 1 and runs as long as the run time
   a. If the fault has not be set and the fault type array is zero, initialize the fault type and fault occurrence variables if
   b. If the fault has not be set and the fault type array is not zero
      i. Add one to the fault set counter
      ii. Check if the fault set counter is equal to one
1. If the previous fault type and current fault type are equal (as the program was programmed so that there was overlap), then set the fault occurrence to the current n and set the fault type to the nth part of the fault type array.

3) If the fault type is not of integer 0 (which means NONE)
   a. Run the fault location algorithm, output the distance generated

4.3.3.3 IMPLEMENTING THE CONTINUOUS ALGORITHM CODE

The continuous algorithm relies on two functions, two of which have been previously defined: faultDetectorFinal and faultDistance. In this code, the program executes the algorithm that was created as a part of this research so that the fault occurrence did not have to be a given value by the user can be found in Appendix A. The following schematic demonstrates the necessary inputs, outputs, and dependencies for the function.
Function: runFaultDetector
Description: To continuously iterate through the iabc array, detect a fault and classify it, and then find its fault location.

**runFaultDetector**
(Model Version)

---

**FIGURE 4.6 SCHEMATIC OF THE runFaultDetector CODE**
4.4 ADJUSTING THE CODE FOR THE HDL CONVERTER

The runFaultDetector code has been introduced in the previous sections of this chapter, as well as tested and verified in Chapter 5. Thus, the next important step in the algorithm and code implementation is to create a Simulink model such that Simulink itself can easily convert the research-developed code to HDL code. The importance in doing this is to prove that the runFaultDetector function, as defined in Appendix A, can freely run on an FPGA chip.

In order to make a Simulink model out of code, one must use a MATLAB Function block in the Simulink Library, such that the runFaultDetector code can be copied and pasted into the newly-created Simulink model so that it can be generated. There were several steps that had to be taken before creating an HDL Converter-ready model.

The first step was to alter the input values so that the iabc and vabc arrays were no longer of any use to the function. In a real-world application of an FPGA implementation, the Ia, Ib, Ic, Va, Vb, and Vc values could be read in by the methods outlined in Chapter 3. With that in mind, the iabc and vabc variables were altered to Ia, Ib, Ic, aVa, Vb, and Vc.

Next, there had to be counter generated so that an array could be formed to generate the new iabc and vabc data for testing. The array size would be defined by the iterator length, which would be found with (27).

\[
\text{Iterator} = \text{runtime} \times \text{samplesPerCycle}
\]  

(27)
Thus, a while-loop was generated to fill the array with every new input value, and do a continuous read until the array was full. This way, the array could be run through the rest of the original runFaultDetector code.

Next, all output values had to be initialized before a new if statement, which was generated in the while-loop. This if-statement ensured that as long as the iterator was less than the total array length, then the output values would not be real, as they would be set to 0. This way, no misleading data could be presented to the user. Once the array was filled, the continueWithProgram variable would be set to 1, implying in binary that it is true, and continue with the original runFaultDetector code.

It took a lot of trial and error, as well as learning the demands of Simulink to finally settle with this overly simplified, yet perfectly adequate final product. The research conducted included using timing delays, and a lot of Simulink blocks which are not listed in the final product of this piece, however, the final code that was able to run in the MATLAB Function block has been placed in Appendix C, named ‘simulateFaultDetector’.

Now that the model has been introduced, it is first important to introduce the schematic, as the code in this chapter have been presented.
Function: simulateFaultDetector
Program purpose: To continuously run every 15/60 seconds such that it can detect a fault and specify the fault location on the line.

---

**TOP LEVEL DESIGN**

(Model Design)

---

**FIGURE 4.7 SCHEMATIC OF THE TOP LEVEL DESIGN FOR THE MODEL CODE**
The research conducted in order to find the best model for this much-needed solution kept in mind that the parameters that were generated from the SimPowerSystem model had to be manually placed as constant values, keeping in mind that in a real-world application, these values would also have to be manually placed as well. Finally, six input ports and three output ports were placed in the model, resembling the 6 constantly read input values: Va, Vb, Vc, Ia, Ib, and Ic, and the output values: faultType (integer), faultOcc(integer), and faultDist(integer). The model generated for this research is the following.

**Figure 4.8 Simulink version of the research-based algorithm**
4.5 IMPLEMENTING THE HDL CONVERTER

The HDL converter is a plug-in that can be initialized in Simulink, and depending on how much code-development ones does on their computer, there is a Microsoft Service Pack that must be installed in order for the HDL converter to run. After this, the Configuration Parameters had to be altered to only running in Variable-Step Discrete, which prevented a translation error from occurring. Once these steps have been taken, and the Simulink model itself runs without any errors, then one must run the HDL Workflow Advisor to check for specific HDL-converter issues. In the research that was conducted, there were several unnecessary errors that occurred, along with the HDL Converter tool not allowing certain types of functionality to occur.

4.5.1 RUNNING THE HDL WORKFLOW ADVISOR

The following steps were crucial to ensuring that all possible errors for the HDL converter were avoided. The VHDL and Verilog program that this research would use to continue the HDL code to be developed further is Xilinx 13.4. Thus, the MATLAB Toolpath must be set in the following manner by using this command in the MATLAB Command-Line window.

```matlab
hdlssetuptoolpath('ToolName','Xilinx ISE','ToolPath','C:\Xilinx\13.4\ISE_DS\ISE\bin\nt64\ise.exe');
```
Next, the HDL Workflow Adviser may now be opened with the model name selected.

![System Selector](image)

**FIGURE 4.9 SNAPSHOT OF SELECTING THE SYSTEM TO CONVERT**

Since the toolpath has been set, then the Synthesis Tool in the Set Target Device and Synthesis tool can select Xilinx, where FPGA device and its parameters were selected as the following.

- **Family**: Spartan 6
- **Device**: xc6slx45t
- **Package**: fgg 484
- **Speed**: -3

Below is a screenshot of the verification that this works.
Finally, all possible compatibility checks were run and corrected within the model and the code, where the code in Appendix C, as well as the model shown in Fig 4.10 will pass all verification steps.

**Figure 4.10 Snapshot of setting the FPGA parameters**

**Figure 4.11 Snapshot confirming that the system is convertible**
However, through all of this effort of perfecting the algorithms and making sure the model is compliant with the HDL Converter Tool, there seems is a bug in the converter itself, as the following dialogue box would continuously run and never stop, yielding that the error is in the converter itself, and not in the code that has been developed.

![Figure 4.12 Snapshot of HDL Conversion Errors](image)

**Figure 4.12 Snapshot of HDL Conversion Errors**

While the ideal case was to generate HDL Code such that it could be implemented on the Spartan-6 FPGA chip specified, passing all of the verification tests proves the functionality that the research demanded: for it to be possible that the algorithms, the research-based algorithm, and then the model which simulated the algorithm could be
prepared to function properly on an FPGA chip, and can in the future, given that MATLAB develops a fix for this bug.

4.6 OVERVIEW OF TESTING PROCESS

The first algorithm introduced in this thesis is tested by running the fault occurrence at three different points out of the entire list of points in the cycle: the first-most possible fault occurrence, the middle-most fault occurrence, and the last-most fault occurrence. The verification of all possible fault types as well as the occurrence values rely in the graphs generated.

The second algorithm in this thesis was tested by using the same ratio of Line 1 to Line 2 block, where the entirety of the line is 200km, and three test cases at varying ratios occurred to prove that the per unit output was correct. This, all possible fault types are tested with their real-world resistance values.

The testing process for the research-based algorithm implemented was to run the first algorithm test and verification procedure. If the fault detector detects the correct fault with relative accuracy, then this algorithm operates correctly.
CHAPTER FIVE: EXPERIMENTAL RESULTS

5.1 INTRODUCTION TO TESTING PROCESS

The research which has been conducted is, in part, a code based version and implementation of two fundamental algorithms: fault type detection and classification and fault location. In turn, the fundamental point of the research is not to prove that the algorithms work, but to prove that the implementation of the algorithms is valid. Due to certain real-world characteristics that have also affected the data that has been simulated, then the algorithms themselves are also not foolproof. In the following display of experimental results, the constructions of the algorithms are also introduced.

5.2 FAULT TYPE DETECTION, CLASSIFICATION, AND LOCATION TESTING STRATEGY

From a top-down-approach, it is crucial to identify how the two algorithms to be tested were tested with the algorithm that was developed as a part of this research. The continuously running algorithm must have also been tested to ensure accuracy and precision. In that, a lot of errors presented themselves when accessing the non-fool-proof algorithms for this design. In order to ensure that the continuously running algorithm worked properly, all possible fault types, occurrences, and locations had to be run through the simulation. In short, the input data from the SimPowerSystems had to at least be changed more than once to prove that research-developed algorithm and the
implemented algorithms themselves ran properly. Thus, the following methodology was created to ensure accuracy of the research-based algorithm.

5.2.1 CHANGING THE FAULT OCCURRENCE

Since the runtime could be specified by the user as an input to the top-level module, then the runtime was set such that it was long enough to test the first possible occurrence, the middle-most possible occurrence, and the last possible occurrence. Thus, the run time was set to 15/60, so that the first possible occurrence was 2/60, the middle-most was 6/60, and the last was 10/60. The purpose of testing this was to test the research-developed algorithm for accuracy, though it has to be run with the idea in mind that the last three elements of the array (if there were to be a fault on the line) would be discarded, though this issue is compensated for in the HDL conversion code.

The only way to set the actual fault occurrence was by changing the parameter in the SimPowerSystems model, so before running the test simulations, the fault occurrence had to be set in the Three-Phase-Fault block, in the Transition Time(s) parameter. For simulation purposes, that parameter could be redefined as the following.

Transition Time(s) = [faultOccurrence 1]

This means that once the fault has occurred, it continues until the end of the simulation, at 1. So, the Transition Time(s) parameter was changed to the following.

Transition Time(s) = [2/60 1]
Transition Time(s) = [6/60 1]
Transition Time(s) = [10/60 1]
The fault occurrence values only matter when testing the ability of the research-driven algorithm, as well as the fault detection and classification algorithm, which fundamentally aids the research algorithm. Thus, when testing the fault location algorithm, the fault occurrence can be set to a constant value, which was decided to be 2/60.

5.2.2 CHANGING THE FAULT TYPE

In each case of the transition time, the fault type had to also be specified in the SimPowerSystems model. In the Three-Phase-Fault block, there are check boxes for each line one would want to specify, such as the following.

- Phase A fault
- Phase B fault
- Phase C fault
- Ground fault

Thus, in order for a fault to be simulated, two or more boxes had to have been checked to represent the following possible fault types.

- Phase-to-phase: AB, BC, CA
- Single-Phase-to-ground: AG, BG, CG
- Double-Phase-to-ground: ABG, BCG, CAG
- Three-Phase: ABC
  (Three-Phase-to-Ground: ABCG)

The three-phase-to-ground possibility was not tested since the SimPowerSystem model that was used is not ungrounded, as that is the only way for this fault type to occur. It can
be assumed that if the rest of the fault types have been detected accurately, then the ABCG fault would also be detected if the system were to be ungrounded.

This section is crucial in terms of testing all algorithms, as the fault type must be set, detected, and then used in order to use the right equation for determining the fault distance.

5.2.3 CHANGING THE RESISTANCE VALUES

Finally, the last parameters to alter in order to simulate a fault in the SimPowerSystems model that affect the testing of the algorithms are the Fault resistances Ron (Ohms) and Ground Resistances Rg (Ohms). In order to test the fault detection and classification algorithm, these parameters were set to a constant value, which are the following.

Fault resistances Ron (Ohms) = 0.01

Ground Resistances Rg (Ohms) = 5

In order to test the fault location algorithms, these parameters were set to the following values, based on what type of fault would be occurring.

Phase-to-phase:

Ron = 0.5,5

Rg = N/A

Single-Phase-to-ground:

Ron = 0.01
5.3 TESTING THE RESEARCH-BASED AND FAULT DETECTION AND CLASSIFICATION ALGORITHMS

The parameters, as explained above in sections 5.2.1 through 5.2.3, have been altered to demonstrate that the research-based and fault-type detection and classification algorithms function properly. Thus, in order to prove this, graphs were created to output, as the title, as well as for a visual example in the waveforms, that the fault type and fault occurrence are the same as the parameters that were set in the SimPowerSystems model.

Due to the simulation parameters of the fault, in some cases, the fault occurrence would be at most +2/60 from what had originally been set, but never less than the original fault setting. This is strictly an issue with the research-developed algorithm, as there is no clean way to be 100% accurate with when the fault occurred after it has happened. Since the timing delay is not an issue in both the real world (as it takes longer than 2/60 to
correct), as well as for the SimPowerSystems model whose transition time goes to 1, then the output of the fault occurrence does not matter. In terms of the fault location algorithm, the values after the fault has occurred, regardless of which cycle, should still yield a valid fundamental phasor for all Ia, Ib, Ic, Va, Vb, and Vc so that the location algorithm can be implemented.

The code and its sub-functions used to simulate all of the algorithms can be found in Appendix A, with the function name of “simAll.” The schematics for testing these algorithms are below, where the lower levels are the same as introduced in Chapter 4.
Function Name: simAll

Program purpose: To simulate the SimPowerSystems model, and the research based algorithm which simulates the fault detection and classification algorithms

**TOP LEVEL DESIGN**
(Simulation Design)

---

**FIGURE 5.1 SCHEMATIC FOR THE TOP LEVEL DESIGN FOR THE SIMULATION CODE**

---

56
**Function Name:** simulateFaultDetector

**Program purpose:** To continuously run every 15/60 seconds such that it can detect a fault and specify the fault location on the line.

---

**Figure 5.2 Schematic for the Simulation Code for runFaultDetector**
Thus, the verification of each occurrence as well as each fault type is presented as follows. Note: the waveform figure x-axis, Time, is in seconds.

Fault Type: AG, Detected Fault Type: AG
Set Fault Occurrence: 2/60, Detected Fault Occurrence: 2/60

![Figure 5.3 Fault Type AG at 2/60](image1)

**Figure 5.3 Fault Type AG at 2/60**

Fault Type: AG, Detected Fault Type: AG
Set Fault Occurrence: 6/60, Detected Fault Occurrence: 6/60

![Figure 5.4 Fault Type AG at 6/60](image2)

**Figure 5.4 Fault Type AG at 6/60**
Fault Type: AG, Detected Fault Type: AG

Set Fault Occurrence: 10/60, Detected Fault Occurrence: 10/60

![Figure 5.5 Fault Type AG at 10/60](image)

Fault Type: BG, Detected Fault Type: BG

Set Fault Occurrence: 2/60, Detected Fault Occurrence: 3/60

![Figure 5.6 Fault Type BG at 2/60](image)
Fault Type: BG, Detected Fault Type: BG

Set Fault Occurrence: 6/60, Detected Fault Occurrence: 7/60

**Figure 5.7** Fault Type BG at 6/60

Fault Type: BG, Detected Fault Type: BG

Set Fault Occurrence: 10/60, Detected Fault Occurrence: 11/60

**Figure 5.8** Fault Type BG at 10/60
Fault Type: CG, Detected Fault Type: CG

Set Fault Occurrence: 2/60, Detected Fault Occurrence: 4/60

Figure 5.9 Fault Type CG at 2/60

Fault Type: CG, Detected Fault Type: CG

Set Fault Occurrence: 6/60, Detected Fault Occurrence: 8/60

Figure 5.10 Fault Type CG at 6/60
Fault Type: CG, Detected Fault Type: CG

Set Fault Occurrence: 10/60, Detected Fault Occurrence: 11/60

**Figure 5.11 Fault Type CG at 10/60**

Fault Type: ABG, Detected Fault Type: ABG

Set Fault Occurrence: 2/60, Detected Fault Occurrence: 2/60

**Figure 5.12 Fault Type ABG at 2/60**
Fault Type: ABG, Detected Fault Type: ABG
Set Fault Occurrence: 6/60, Detected Fault Occurrence: 6/60

FIGURE 5.13 FAULT TYPE ABG AT 6/60

Fault Type: ABG, Detected Fault Type: ABG
Set Fault Occurrence: 10/60, Detected Fault Occurrence: 10/60

FIGURE 5.14 FAULT TYPE ABG AT 10/60
Fault Type: BCG, Detected Fault Type: BCG

Set Fault Occurrence: 2/60, Detected Fault Occurrence: 2/60

Figure 5.15 Fault Type BCG at 2/60

Fault Type: BCG, Detected Fault Type: BCG

Set Fault Occurrence: 6/60, Detected Fault Occurrence: 6/60

Figure 5.16 Fault Type BCG at 6/60
Fault Type: BCG, Detected Fault Type: BCG

Set Fault Occurrence: 10/60, Detected Fault Occurrence: 10/60

FIGURE 5.17 FAULT TYPE BCG AT 10/60

Fault Type: CAG, Detected Fault Type: CAG

Set Fault Occurrence: 2/60, Detected Fault Occurrence: 2/60

FIGURE 5.18 FAULT TYPE CAG AT 2/60
Fault Type: CAG, Detected Fault Type: CAG

Set Fault Occurrence: 6/60, Detected Fault Occurrence: 6/60

**Figure 5.19 Fault Type CAG at 6/60**

Fault Type: CAG, Detected Fault Type: CAG

Set Fault Occurrence: 10/60, Detected Fault Occurrence: 10/60

**Figure 5.20 Fault Type CAG at 10/60**
Fault Type: AB, Detected Fault Type: AB

Set Fault Occurrence: 2/60, Detected Fault Occurrence: 2/60

Figure 5.21 Fault Type AB at 2/60

Fault Type: AB, Detected Fault Type: AB

Set Fault Occurrence: 6/60, Detected Fault Occurrence: 6/60

Figure 5.22 Fault Type AB at 6/60
Fault Type: AB, Detected Fault Type: AB
Set Fault Occurrence: 10/60, Detected Fault Occurrence: 10/60

![Fault Type AB at 10/60](image)

**Figure 5.23 Fault Type AB at 10/60**

Fault Type: BC, Detected Fault Type: BC
Set Fault Occurrence: 2/60, Detected Fault Occurrence: 3/60

![Fault Type BC at 2/60](image)

**Figure 5.24 Fault Type BC at 2/60**
Fault Type: BC, Detected Fault Type: BC

Set Fault Occurrence: 6/60, Detected Fault Occurrence: 7/60

FIGURE 5.25 FAULT TYPE BC AT 6/60

Fault Type: BC, Detected Fault Type: BC

Set Fault Occurrence: 10/60, Detected Fault Occurrence: 11/60

FIGURE 5.26 FAULT TYPE BC AT 10/60
Fault Type: CA, Detected Fault Type: CA
Set Fault Occurrence: 2/60, Detected Fault Occurrence: 3/60

FIGURE 5.27 FAULT TYPE CA AT 2/60

Fault Type: CA, Detected Fault Type: CA
Set Fault Occurrence: 6/60, Detected Fault Occurrence: 7/60

FIGURE 5.28 FAULT TYPE CA AT 6/60

70
Fault Type: CA, Detected Fault Type: CA
Set Fault Occurrence: 10/60, Detected Fault Occurrence: 11/60

![Diagram of Fault Type CA at 10/60]

**Figure 5.29 Fault Type CA at 10/60**

Fault Type: ABC, Detected Fault Type: ABC
Set Fault Occurrence: 2/60, Detected Fault Occurrence: 2/60

![Diagram of Fault Type ABC at 2/60]

**Figure 5.30 Fault Type ABC at 2/60**
Fault Type: ABC, Detected Fault Type: ABC

Set Fault Occurrence: 6/60, Detected Fault Occurrence: 6/60

Figure 5.31 Fault Type ABC at 6/60

Fault Type: ABC, Detected Fault Type: ABC

Set Fault Occurrence: 10/60, Detected Fault Occurrence: 10/60

Figure 5.32 Fault Type CG at 10/60
5.4 TESTING THE FAULT LOCATION

The fault location algorithm relies on certain parameters being set, such as those specified in section 5.2. In that section, it was stated that the fault location algorithm would be tested using the fault occurrence of 2/60 while the ground and fault impedances change as specified in 5.2.3. These ground and fault location alterations were run using two different line length parameters, to prove the precision is the same, regardless of the line length or the input values. In the following result section, the Line 1 and Line 2 blocks were altered in the following way.

Simulation 1

Line 1 = 50km
Line 2 = 150km

Simulation 2

Line 1 = 100 km
Line 2 = 100 km

Simulation 3

Line 1 = 150 km
Line 2 = 50 km

As explained before, the output of the fault location algorithm for testing purposes is in per unit since the SimPowerSystems model outputs vectors iabc and vabc in per unit as well. However, in the HDL conversion code, the per unit conversion does not exist, and assumes that the Z0 and Z1 parameters have already been converted such that they match the iabc and vabc respective units. Since the outputs are in per unit, then the per unit...
values for each fault type should remain relatively similar to one another, dependent on the fault type classification. For Simulation 1, the ratio is 50/200, so the values should be around .25. For simulation 2, the ratio is 100/200, so the values should be around .50. For simulation 3, the ratio is 150/200, so the values should be around .75.

5.4.1 FAULT LOCATION SIMULATION 1 VERSUS SIMULATION 2 RESULTS

The simulation results below represent what the fault location algorithm generated based on the after fault data that was fed to it. Since the fault location algorithm is a close estimation based on the impedance values, the examination of the results yields two particular noteworthy pieces of information.

First, that the fault length does alter the frequency by which the fault is readily detected.

Second, it can be concluded that the ground fault input points highly affected the Sim 1 data, while in all cases, of Sim 1, Sim 2, and Sim3, the initial case of 5 Ohms for the ground fault, or the fault occurrence of 0.05 yielded an extremely similar value to the suggested ratios for each simulation.

The following tables represent the fault location simulation and test results.
### Phase-to-Ground Sim 1

#### Table 5.1 Phase to Ground Simulation 1

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Rg (Ohms)</th>
<th>Rn (Ohms)</th>
<th>Distance (p.u.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG</td>
<td>5</td>
<td>0.01</td>
<td>0.2744</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.4325</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.6814</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.9453</td>
</tr>
<tr>
<td>BG</td>
<td>5</td>
<td>0.01</td>
<td>0.2743</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.4325</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.6814</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.9453</td>
</tr>
<tr>
<td>CG</td>
<td>5</td>
<td>0.01</td>
<td>0.2743</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.4324</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.6813</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.9453</td>
</tr>
</tbody>
</table>

### Phase-to-Ground Sim 2

#### Table 5.2 Phase to Ground Simulation 2

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Rg (Ohms)</th>
<th>Rn (Ohms)</th>
<th>Distance (p.u.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG</td>
<td>5</td>
<td>0.01</td>
<td>0.5321</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.7216</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>1.0344</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>1.3774</td>
</tr>
<tr>
<td>BG</td>
<td>5</td>
<td>0.01</td>
<td>0.532</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.7216</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>1.0344</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>1.3774</td>
</tr>
<tr>
<td>CG</td>
<td>5</td>
<td>0.01</td>
<td>0.532</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.7216</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>1.0344</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>1.3774</td>
</tr>
</tbody>
</table>
Phase-to-Ground Sim 3

Table 5.3 Phase to Ground Simulation 3

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Rg (Ohms)</th>
<th>Rn (Ohms)</th>
<th>Distance (p.u.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG</td>
<td>5</td>
<td>0.01</td>
<td>0.8029</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>1.0989</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>1.5890</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>2.1243</td>
</tr>
<tr>
<td>BG</td>
<td>5</td>
<td>0.01</td>
<td>0.8031</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>1.0988</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>1.5889</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>2.1241</td>
</tr>
<tr>
<td>CG</td>
<td>5</td>
<td>0.01</td>
<td>0.8025</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>1.0987</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>1.5888</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>2.1241</td>
</tr>
</tbody>
</table>

These results for all three simulations yield that the single phase to ground faults are not detectable beyond a ground fault larger than 50 ohms, since all values have a result of something greater than double the desired ratio, thus exhausting the possible location on the line that a fault could occur.
### Double-Phase-to-Ground Sim 1

Table 5.4 Double Phase to Ground Simulation 1

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Rg (Ohms)</th>
<th>Rn (Ohms)</th>
<th>Distance (p.u.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABG</td>
<td>5</td>
<td>0.01</td>
<td>0.2199</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.2145</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.2218</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.2504</td>
</tr>
<tr>
<td>BCG</td>
<td>5</td>
<td>0.01</td>
<td>0.2201</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.2147</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.2219</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.2507</td>
</tr>
<tr>
<td>CAG</td>
<td>5</td>
<td>0.01</td>
<td>0.2391</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.2296</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.2269</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.2278</td>
</tr>
</tbody>
</table>

### Double Phase-to-Ground Sim 2

Table 5.5 Double Phase to Ground Simulation 2

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Rg (Ohms)</th>
<th>Rn (Ohms)</th>
<th>Distance (p.u.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABG</td>
<td>5</td>
<td>0.01</td>
<td>0.5024</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.5027</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.5027</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.5027</td>
</tr>
<tr>
<td>BCG</td>
<td>5</td>
<td>0.01</td>
<td>0.5029</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.5029</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.5031</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.5029</td>
</tr>
<tr>
<td>CAG</td>
<td>5</td>
<td>0.01</td>
<td>0.5031</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.5033</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.5031</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.5034</td>
</tr>
</tbody>
</table>
Double Phase-to-Ground Sim 3

Table 5.6 Phase to Phase Simulation 3

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Rg (Ohms)</th>
<th>Rn (Ohms)</th>
<th>Distance (p.u.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABG</td>
<td>5</td>
<td>0.01</td>
<td>0.7072</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.6894</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.7597</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.7597</td>
</tr>
<tr>
<td>BCG</td>
<td>5</td>
<td>0.01</td>
<td>0.7089</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.6911</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.7616</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.7616</td>
</tr>
<tr>
<td>CAG</td>
<td>5</td>
<td>0.01</td>
<td>0.8203</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.01</td>
<td>0.7876</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>0.01</td>
<td>0.7750</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>0.01</td>
<td>0.7753</td>
</tr>
</tbody>
</table>

The results for the Double-Phase-to-Ground faults yield that they result hovers around the desired per unit, thus it all possible faults would be detectable in the Rg range of less than 75 ohms, where 75 is a realistic real-world value.
### Phase-to-Phase Sim 1

**Table 5.7 Phase-to-Phase Simulation 1**

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Rg (Ohms)</th>
<th>Rn (Ohms)</th>
<th>Distance (p.u.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>N/A</td>
<td>0.5</td>
<td>0.2507</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>5</td>
<td>0.3055</td>
</tr>
<tr>
<td>BC</td>
<td>N/A</td>
<td>0.5</td>
<td>0.2509</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>5</td>
<td>0.3055</td>
</tr>
<tr>
<td>CA</td>
<td>N/A</td>
<td>0.5</td>
<td>0.2509</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>5</td>
<td>0.3055</td>
</tr>
</tbody>
</table>

### Phase-to-Phase Sim 2

**Table 5.8 Phase-to-Phase Simulation 2**

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Rg (Ohms)</th>
<th>Rn (Ohms)</th>
<th>Distance (p.u.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>N/A</td>
<td>0.5</td>
<td>0.5031</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>5</td>
<td>0.5633</td>
</tr>
<tr>
<td>BC</td>
<td>N/A</td>
<td>0.5</td>
<td>0.5035</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>5</td>
<td>0.5633</td>
</tr>
<tr>
<td>CA</td>
<td>N/A</td>
<td>0.5</td>
<td>0.5035</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>5</td>
<td>0.5633</td>
</tr>
</tbody>
</table>
Phase-to-Phase Sim 3

Table 5.9 Phase to Phase Simulation 3

<table>
<thead>
<tr>
<th>Fault Type</th>
<th>Rg (Ohms)</th>
<th>Rn (Ohms)</th>
<th>Distance (p.u.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>N/A</td>
<td>0.5</td>
<td>0.7601</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>5</td>
<td>0.8341</td>
</tr>
<tr>
<td>BC</td>
<td>N/A</td>
<td>0.5</td>
<td>0.7600</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>5</td>
<td>0.8339</td>
</tr>
<tr>
<td>CA</td>
<td>N/A</td>
<td>0.5</td>
<td>0.7601</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>5</td>
<td>0.8339</td>
</tr>
</tbody>
</table>

The results for the Phase-to-Phase faults yield that they result hovers around the desired per unit, thus it all possible faults would be detectable in the Ron range of less than 5 ohms, where 5 is a realistic real-world value.
The results for the Three-Phase faults yield that they result hovers around the desired per unit, thus it all possible faults would be detectable in the Ron range of less than 5 ohms, where 5 is a realistic real-world value.
CHAPTER SIX: CONCLUSIONS AND SCOPE FOR FUTURE WORK

This thesis attempted to create the best possible ways in which fault detection and classification as well as location algorithms could be implemented in a singular function. Each algorithm is needed in the final product, as without the phase type or the fault occurrence detected by the specialized version of the first algorithm presented, then the fault location algorithm would not have worked at all.

The most realistic approaches were made when creating test sets, as the importance of ensuring that the research-based algorithm for sifting through the data array which was fed in, was to attempt to access three different areas in the fault occurrence, thus ensuring that the fault detector would detect a fault at the first sign of a fault on the line. Every fault type was tested for all algorithms, thus ensuring that the fault type detection algorithm aided in the rest of the design. Testing for the fault located proved that the ratio yielded relatively similar results and trends in the outputs aided in proving that the algorithm itself was good enough to be translated to HDL code.

While the HDL code conversion did not fully convert all of the code, a whole lot of time was spent in perfecting the algorithm that would be run in the HDL converter. Even though there were a lot of setbacks, the future scope of work looks bright. Thus, as a part of the future scope of work, some important conclusions were drawn as well.

Conclusion: The Matlab HDL Converter was not able to completely convert the code, and it should have this functionality built in, without bugs.

- Future Work: Wait on MATLAB to develop a better HDL converter, or write the Verilog code for new research
Conclusion: Though the fault detection algorithm works, the minimum number of cycles to make it run properly is four. This might not be the most effective in the long run.

- Future Work: Attempt to make the code such that when writing for the power system module that could be converted into HDL, ensure that the 5 previous cycles are automatically stored to the beginning of the next set to ensure that no data is lost. Wait on HDL Converter to work, such that a storage device can be implemented. This way no data will be lost.
function [faultType, faultOcc, estFaultDistance] = runFaultDetector(i_abc,v_abc,Z0,Z1,runTime,totalRunTime)

%#codegen

{%
Engineer: Christina Yeoman
Adviser: Dr. Yuan Liao
Program purpose: To continuously iterate through the iabc array, detect a fault and classify it, and then find its fault location.

Program dependencies: faultDetectorFinal, faultDistance

Standard inputs:
i_abc = iabc (from workspace)
v_abc = vabc (from workspace)
runTime = 10/60 to 15/60
faultOcc = from 1/60 to 12/60

Expected outputs:
faultType & faultTypeWord: 0 - NONE; 1 - AG; 2 - BG; 3 - CG; 4 - AB;
5 - BC; 6 - CA; 7 - ABG; 8 - BCG; 9 - ACG; 10 - ABCG;
11 - ABC; 12 - MISMATCH
faultOcc = 1/60 to 30/60
estFaultDistance = 0 to 200 km
%
%

%initializing the parameter needed for the FOR loop
faultSet = 0;
faultType = 0;
faultTypeTemp = 0;
I_cycForLoc = zeros(1,3);
V_cycForLoc = zeros(1,3);
faultOcc = 0;
calculating the run time so that we know how to iterate through
the time cycle properly
runTimeFrac = runTime/totalRunTime;

setting the faultType array to all zeros, which means NONE
it will only be set to something else if there is a fault
assert(runTime<=60);
faultTypeArr = zeros(1,runTime);
faultIforLoc = zeros(runTime,6);
faultVforLoc = zeros(runTime,6);

this FOR loop iterates through the fault detector function and sets it to
the fault type that has occurred, this can be NONE
this FOR loop implements the faultDetectorFinal function
function [faultType] = 
faultDetectorFinal(I_abc,In_freq,samPerCycle,numCycleInWnd,runTime,faultOcc)
for m=1:runTime
    faultOccIter = m/totalRunTime;
    if(m < (runTime-3))
        [faultTypeOut,I_cycForLoc,V_cycForLoc] =
faultDetectorFinal(I_abc,v_abc,60,128,1,runTimeFrac,faultOccIter);
        faultTypeArr(m) = faultTypeOut;
        faultIforLoc(m,:) = I_cycForLoc;
        faultVforLoc(m,:) = V_cycForLoc;
    else
        faultTypeArr(m) = 0;
        faultIforLoc(m,:) = zeros(1,6);
        faultVforLoc(m,:) = zeros(1,6);
    end
end

this FOR loop iterates through the faultType array which has been set to
the fault types which have been set throughout the entire run time
it does this by checking if the faultArray was set to anything beyond zero
for n=1:runTime
    if(faultTypeArr(n) == 0 && faultSet == 0)
        faultType = 0;
        faultOcc = 0;
    elseif(faultTypeArr(n) ~= 0)
        faultSet = faultSet + 1;
    end
    if(n > 1)
        if(faultTypeArr(n) == faultTypeArr(n-1))
            faultOcc = n;
        end
    end

faultType = faultTypeArr(n);
end
end
end
end

if(faultType ~= 0)
  %function [estimatedDistance] =
  faultDistance(I_cycForLoc,faultTypeNum,faultOcc,groundResistance,phaseResistance)
    estFaultDistance =
  faultDistance(faultIforLoc(faultOcc,:),faultVforLoc(faultOcc,:),faultType,Z0,Z1);
else
  estFaultDistance = 0;
end
end
function [faultType,I_cycMoment,V_cycMoment] = faultDetectorFinal(I_abc,V_abc,In_freq,samPerCycle,numCycleInWnd,runTime,faultOcc)

%#codegen

%{
Engineer: Christina Yeoman
Adviser: Dr. Yuan Liao
Program purpose: this is a single-run function that uses the faultOcc variable to check if a fault has occurred on the power line.
The fault type is outputted in an integer value so that it may be parsed later. Note that the fault occurrence must be an input so that this algorithm can run properly.

Standard inputs:
I_abc = iabc (from the SimPowerSystems model)
In_freq = 60
samPerCycle = 128
numCycleInWnd = 1
runTime = 10/60 to 15/60
faultOcc = from 2/60 to 12/60

Expected outputs:
faultType: 0 to 11, where these numbers represent the following:
0 - NONE; 1 - AG; 2 - BG; 3 - CG; 4 - AB;
5 - BC; 6 - CA; 7 - ABG; 8 - BCG; 9 - ACG; 10 - ABCG;
11 - ABC
I_cycMoment = 1x3 array, values 0 to 1.
V_cycMoment = 1x3 array, values 0 to 1.

Function Dependencies:
performFFTFinal, faultDetectorI
%
}
%---finding the size of iabc and vabc, the continuous
%voltage and current representations
%of the simulink model for this fault detection
samFreq = In_freq*samPerCycle;
runTimeScaled = runTime*samFreq;

%defining the first element in each array as well as going back one
%whole cycle so that the before fault variables can be found
startPoint = 1;
beforeFault = (faultOcc*60*samPerCycle - samPerCycle) + startPoint;

%defining the first element in each array as well as going forward two
%whole cycles so that the after fault variables can be found
afterFault = (faultOcc*60*samPerCycle + (2*samPerCycle)) + startPoint;

%-------performing fft calculations for before and after fault-------
%fft code performs the fast fourier transform and outputs the magnitude
%of what it is fed in this case, it is run to find the current and
%voltage signals for magnitude and phase angle of phases a, b, and c

%finding the current values for the fault location and detection
%algorithms
%this uses the performFFTFinal function
%function [mag,ang] =
performFFTFinal(startTime,compBegPoint,In_freq,samPerCycle,numCycleInWnd,runTime,sigPassed)
[la_mag,la_ang] =
performFFTFinal(startPoint,afterFault,In_freq,samPerCycle,numCycleInWnd,runTimeScaled,I_abc(:,1));
[lan_mag,lan_ang] =
performFFTFinal(startPoint,beforeFault,In_freq,samPerCycle,numCycleInWnd,runTimeScaled,I_abc(:,1));
[lb_mag,lb_ang] =
performFFTFinal(startPoint,afterFault,In_freq,samPerCycle,numCycleInWnd,runTimeScaled,I_abc(:,2));
[lbn_mag,lbn_ang] =
performFFTFinal(startPoint,beforeFault,In_freq,samPerCycle,numCycleInWnd,runTimeScaled,I_abc(:,2));
[lc_mag,lc_ang] =
performFFTFinal(startPoint,afterFault,In_freq,samPerCycle,numCycleInWnd,runTimeScaled,I_abc(:,3));
[lc_mag,lc_ang] =
performFFTFinal(startPoint,beforeFault,In_freq,samPerCycle,numCycleInWnd,runTimeScaled,I_abc(:,3));
\[ [I_{cn\_mag}, I_{cn\_ang}] = \text{perform\_FFT\_Final}(\text{startPoint, beforeFault, In\_freq, samPerCycle, numCycleInWnd, runTimeScaled, I\_abc(:,3))}; \]

\%
\%finding the voltage values for performing the fault location algorithm
\[
[\text{Va\_mag}, \text{Va\_ang}] = \text{perform\_FFT\_Final}(\text{startPoint, afterFault, In\_freq, samPerCycle, numCycleInWnd, runTimeScaled, V\_abc(:,1))};
\]
\[
[\text{Vb\_mag}, \text{Vb\_ang}] = \text{perform\_FFT\_Final}(\text{startPoint, afterFault, In\_freq, samPerCycle, numCycleInWnd, runTimeScaled, V\_abc(:,2))};
\]
\[
[\text{Vc\_mag}, \text{Vc\_ang}] = \text{perform\_FFT\_Final}(\text{startPoint, afterFault, In\_freq, samPerCycle, numCycleInWnd, runTimeScaled, V\_abc(:,3))};
\]

\%
\------finding the fundamental phasor for voltage and current------
\%
\%this is the second element in the magnitude in the magnitude and phase
\%
\%arrays that come out of the fft function, which is the fundamental
\%
\%phasor
\[
\text{Ia} = \text{Ia\_mag}(2) * \exp(1i*\text{Ia\_ang}(2));
\]
\[
\text{Ib} = \text{Ib\_mag}(2) * \exp(1i*\text{Ib\_ang}(2));
\]
\[
\text{Ic} = \text{Ic\_mag}(2) * \exp(1i*\text{Ic\_ang}(2));
\]
\[
\text{Ian} = \text{Ian\_mag}(2) * \exp(1i*\text{Ian\_ang}(2));
\]
\[
\text{Ibn} = \text{Ibn\_mag}(2) * \exp(1i*\text{Ibn\_ang}(2));
\]
\[
\text{Icn} = \text{Icn\_mag}(2) * \exp(1i*\text{Icn\_ang}(2));
\]
\[
\text{I0} = (\text{Ia} + \text{Ib} + \text{Ic})/3;
\]
\[
\text{I\_cycMoment} = [\text{Ia} \text{ Ib} \text{ Ic}];
\]

\%
\%finding the voltage values for performing the fault location algorithm
\[
\text{Va} = \text{Va\_mag}(2) * \exp(1i*\text{Va\_ang}(2));
\]
\[
\text{Vb} = \text{Vb\_mag}(2) * \exp(1i*\text{Vb\_ang}(2));
\]
\[
\text{Vc} = \text{Vc\_mag}(2) * \exp(1i*\text{Vc\_ang}(2));
\]
\[
\text{V\_cycMoment} = [\text{Va} \text{ Vb} \text{ Vc}];
\]

\%
\%------finding the current comparison voltages------
\%
\%defining the comparison variables for current comparison
\[
\text{currentEpsilon1} = 0.3*\text{abs}(\text{Ian});
\]
\[
\text{currentEpsilon2} = 1.4;
\]
\[
\text{currentEpsilon3} = 0.1;
\]
% finding the change in current phase from before to after the fault
deltaIa = Ia - Ian;
deltaIb = Ib - Ibn;
deltaIc = Ic - Icn;

% finding the final current variables
currentDa = abs(deltaIa);
currentDb = abs(deltaIb);
currentDc = abs(deltaIc);
currentD0 = abs(I0);

%-------grouping the current comparison variables-------
currentCompArray = [currentDa,currentDb,currentDc,currentD0;
currentEpsilon1 currentEpsilon2 currentEpsilon3 0];

%---calling the comparison algorithm---------
% output is the fault type based on current comparison variables

% this uses the faultDetectorI function
% function [faultTypeI] = faultDetectorI(currentCompArray)
faultType = faultDetectorI(currentCompArray);

end
Appendix A continued - Continuously-Running Algorithm Code Implementation of Fault Detection, Classification, and Location Code

function [mag,ang] = performFFTFinal(startTime,compBegPoint,In_freq,samPerCycle,numCycleInWnd,runTime,sigPassed)
  %#codegen

  % Engine: Christina Yeoman
  % Adviser: Dr. Yuan Liao
  % Program purpose: This performs the Fast-Fourier Transform as described in section 4.3.1.1 of the thesis document. It is used to create the fundamental phasor’s magnitude and voltage based off of an entire cycle of data.

  % Standard inputs:
  startTime = 1
  compBegPoint = 1 to ((runtime*samPerCycle) – samPerCycle)
  In_freq = 60
  samPerCycle = 128
  numCycleInWnd = 1
  runTime = 10/60 to 15/60
  sigPassed = iabc or vabc from faultDetectorFinal

  % Expected outputs:
  mag: positive value, 0 to fundamental phasor magnitude of the data
  ang: positive value, 0 to fundamental phasor angle of the data

  % Function Dependencies:
  None

  %the length of the entire sample frequency
  samFreq = In_freq * samPerCycle;

  % to prevent errors, must check and make sure that this program is not
%accessing a portion of the array that does not exist
if((compBegPoint + samPerCycle) < runTime)

%if we are trying to get the prevault information
if(startTime==compBegPoint)
  %defining the first full cycle of the signal
  sig = sigPassed(startTime : samFreq/In_freq * numCycleInWnd);
  fftResults = fft(sig);

  %figuring out the magnitude and the angle
  mag = abs(fftResults) * 2/(samPerCycle * numCycleInWnd) / sqrt(2);
  ang = angle(fftResults);

%if we are trying to get the during fault information
else
  %defining the third full cycle of the signal
  sig = sigPassed((compBegPoint + startTime) : ((samFreq/In_freq * numCycleInWnd)+ compBegPoint));
  fftResults = fft(sig);

  %figuring out the magnitude and the angle
  mag = abs(fftResults) * 2/(samPerCycle * numCycleInWnd) / sqrt(2);
  ang = angle(fftResults);
end

%if the code is accessing the array beyond where it was
%available then say both the magnitude and angle are zero
else
  mag = 0;
  ang = 0;
end

end
Appendix A continued - Continuously-Running Algorithm Code Implementation of Fault Detection, Classification, and Location Cod

function [faultTypeI] = faultDetectorI(currentCompArray)

%#codegen

%
Engineer: Christina Yeoman
Adviser: Dr. Yuan Liao
Program purpose: Runs the algorithm specified in section 4.3.1.2, which has been developed by W. Fan and Y. Liao[1]. It uses comparators and the threshold values to yield the fault type on the line if there is one. If not, it will output a number signaling that there wasn’t a fault.

Standard inputs:
    currentCompArray = [currentDa,currentDb,currentDc,currentD0; currentEpsilon1 currentEpsilon2 currentEpsilon3 0];
    All of these values have been defined in the faultDetectorFinal function.
Expected outputs:
    faultType: 0 to 11, where these numbers represent the following:
      0 - NONE; 1 - AG; 2 - BG; 3 - CG; 4 - AB;
      5 - BC; 6 - CA; 7 - ABG; 8 - BCG; 9 - ACG; 10 - ABCG;
      11 – ABC

Function Dependencies:
    None

%

%instantuating the output variable
faultTypeI = 0;

%collecting the data from the input array
currentDa = currentCompArray(1,1);
currentDb = currentCompArray(1,2);
currentDc = currentCompArray(1,3);
currentD0 = currentCompArray(1,4);
currentEpsilon1 = currentCompArray(2,1);
currentEpsilon2 = currentCompArray(2,2);
currentEpsilon3 = currentCompArray(2,3);

%performing the current comparison algorithm
if(currentDa >= currentEpsilon1 && currentDb >= currentEpsilon1 && currentDc >=
currentEpsilon1)
    if ((currentDa/currentDb) >= currentEpsilon2 && (currentDa/currentDc) >=
currentEpsilon2)
        faultTypeI = 1;
    end
    if ((currentDb/currentDa) >= currentEpsilon2 && (currentDb/currentDc) >=
currentEpsilon2)
        faultTypeI = 2;
    end
    if ((currentDc/currentDa) >= currentEpsilon2 && (currentDc/currentDb) >=
currentEpsilon2)
        faultTypeI = 3;
    end
    if ((currentDa/currentDc) >= currentEpsilon2 && (currentDb/currentDc) >=
currentEpsilon2 && currentD0 >=(currentEpsilon3*currentDa))
        faultTypeI = 7;
    end
    if ((currentDa/currentDb) >= currentEpsilon2 && (currentDc/currentDb) >=
currentEpsilon2 && currentD0 <(currentEpsilon3*currentDa))
        faultTypeI = 6;
    end
    if ((currentDb/currentDa) >= currentEpsilon2 && (currentDc/currentDa) >=
currentEpsilon2 && currentD0 >=(currentEpsilon3*currentDb))
        faultTypeI = 8;
    end
    if ((currentDb/currentDa) < currentEpsilon2 && (currentDa/currentDb) <
currentEpsilon2 && (currentDb/currentDc) < currentEpsilon2 && (currentDc/currentDb) <
currentEpsilon2 && (currentDa/currentDc) < currentEpsilon2 && currentD0 < (currentEpsilon3*currentDa))
        faultTypeI = 5;
    end
end

if ((currentDa/currentDb) < currentEpsilon2 && (currentDa/currentDb) <
currentEpsilon2 && (currentDb/currentDc) < currentEpsilon2 && (currentDc/currentDb) <
currentEpsilon2 && (currentDa/currentDc) < currentEpsilon2 && currentD0 < (currentEpsilon3*currentDa))
    faultTypeI = 5;
end

if ((currentDb/currentDa) < currentEpsilon2 && (currentDa/currentDb) <
currentEpsilon2 && (currentDb/currentDc) < currentEpsilon2 && (currentDc/currentDb) <
currentEpsilon2 && (currentDa/currentDc) < currentEpsilon2 && currentD0 < (currentEpsilon3*currentDa))
faultTypeI = 11;
end
if ((currentDb/currentDa) < currentEpsilon2 && (currentDa/currentDb) <
currentEpsilon2 && (currentDb/currentDc) < currentEpsilon2 && (currentDc/currentDb) <
currentEpsilon2 && (currentDa/currentDc) < currentEpsilon2 &&
(currentDa/currentDc) < currentEpsilon2 && currentD0 >= (currentEpsilon3*currentDa))
faultTypeI = 10;
end
else
if (currentDa >= currentEpsilon1 && currentDb < currentEpsilon1 && currentDc <
currentEpsilon1)
faultTypeI = 1;
end
if (currentDa >= currentEpsilon1 && currentDb >= currentEpsilon1 && currentDc <
currentEpsilon1 && (currentDa/currentDb) >= currentEpsilon2)
faultTypeI = 1;
end
if (currentDc >= currentEpsilon1 && currentDa >= currentEpsilon1 && currentDb <
currentEpsilon1 && (currentDa/currentDc) >= currentEpsilon2)
faultTypeI = 1;
end
if (currentDb >= currentEpsilon1 && currentDa < currentEpsilon1 && currentDc <
currentEpsilon1)
faultTypeI = 2;
end
if (currentDa >= currentEpsilon1 && currentDb >= currentEpsilon1 && currentDc <
currentEpsilon1 && (currentDb/currentDa) >= currentEpsilon2)
faultTypeI = 2;
end
if (currentDb >= currentEpsilon1 && currentDc >= currentEpsilon1 && currentDa <
currentEpsilon1 && (currentDb/currentDc) >= currentEpsilon2)
faultTypeI = 2;
end
if (currentDc >= currentEpsilon1 && currentDb < currentEpsilon1 && currentDa <
currentEpsilon1)
faultTypeI = 3;
end
if (currentDb >= currentEpsilon1 && currentDc >= currentEpsilon1 && currentDa <
currentEpsilon1 && (currentDc/currentDb) >= currentEpsilon2)
faultTypeI = 3;
end
if (currentDc >= currentEpsilon1 && currentDa >= currentEpsilon1 && currentDb <
currentEpsilon1 && (currentDc/currentDa) >= currentEpsilon2)
faultTypeI = 3;
end
if (currentDa >= currentEpsilon1 && currentDb >= currentEpsilon1 && currentDc < currentEpsilon1 && (currentDa/currentDb) < currentEpsilon2 && (currentDb/currentDa) < currentEpsilon2 && currentD0 < (currentEpsilon3*currentDa))
    faultTypeI = 4;
end
if (currentDb >= currentEpsilon1 && currentDc >= currentEpsilon1 && currentDa < currentEpsilon1 && (currentDb/currentDc) < currentEpsilon2 && (currentDc/currentDb) < currentEpsilon2 && currentD0 < (currentEpsilon3*currentDb))
    faultTypeI = 5;
end
if (currentDc >= currentEpsilon1 && currentDa >= currentEpsilon1 && currentDb < currentEpsilon1 && (currentDc/currentDa) < currentEpsilon2 && (currentDa/currentDc) < currentEpsilon2 && currentD0 < (currentEpsilon3*currentDc))
    faultTypeI = 6;
end
if (currentDa >= currentEpsilon1 && currentDb >= currentEpsilon1 && currentDc < currentEpsilon1 && (currentDa/currentDb) < currentEpsilon2 && (currentDb/currentDa) < currentEpsilon2 && currentD0 >= (currentEpsilon3*currentDa))
    faultTypeI = 7;
end
if (currentDb >= currentEpsilon1 && currentDc >= currentEpsilon1 && currentDa < currentEpsilon1 && (currentDb/currentDc) < currentEpsilon2 && (currentDc/currentDb) < currentEpsilon2 && currentD0 >= (currentEpsilon3*currentDb))
    faultTypeI = 8;
end
if (currentDc >= currentEpsilon1 && currentDa >= currentEpsilon1 && currentDb < currentEpsilon1 && (currentDc/currentDa) < currentEpsilon2 && (currentDa/currentDc) < currentEpsilon2 && currentD0 >= (currentEpsilon3*currentDc))
    faultTypeI = 9;
end
end
Appendix B

Appendix B continued- Fault Simulation Code (items from Appendix A not included)

function [faultDist] = simAll()

%{
Engineer: Christina Yeoman
Adviser: Dr. Yuan Liao
Program purpose: To simulate the SimPowerSystems model, and the
research based algorithm which simulates the fault detection and
classification algorithms

Program dependencies: runFaultDetector

Standard inputs:
none

Expected outputs:
faultDist = 0 to lineLength
where lineLength is specified in the simPowerSystems model
%} %---- from the SimPowerSystem model

runTime = 15;
totalRunTime = 60;
% 1.6093 is 1 mile in km
% R = R1 R0
% L = L1 L0

R = [0.249168 0.60241]/1.6093;
L = [0.00156277 0.0048303]/1.6093;
lineLength = 200;

% for finding the positive and zero sequence impedance values
jayOmega = (2*pi*60)*1i;

% final positive and zero sequence impedance equations, not converted
Z1_noconv = lineLength*(R(1) + jayOmega*L(1));
Z0_noconv = lineLength*(R(2) + jayOmega*L(2));

% defining the z-base, for testing purposes only
V_base = 500e3;
S_base = 100e6;
Z_base = (V_base^2)/S_base;

% converting these values to z-base, for testing purposes only
Z1 = Z1_noconv/Z_base;
Z0 = Z0_noconv/Z_base;

% simulating the SimPowerSystems model
sim('CY_TwoBusSystemPhasors.mdl');

% running the research-based continuous function
[faultType, faultOcc, faultDist] = runFaultDetector(iabc, vabc, Z0, Z1, runTime, totalRunTime);

% outputting the results, for testing purposes only
% postProcessSim(faultType, faultOcc, iabc);

end
function postProcessSim(faultType, faultOcc, i_abc)

%
Engineer: Christina Yeoman
Adviser: Dr. Yuan Liao
Program purpose: To convert the faultType integer into a string and then run the plotting function

Program dependencies: plotFaultType

Standard inputs:
faultType: 0 to 12, where
  0- NONE; 1 - AG; 2 - BG; 3 - CG; 4 - AB;
  5 - BC; 6 - CA; 7 - ABG; 8 - BCG; 9 - ACG; 10 - ABCG;
  11 - ABC;
faultOcc = 2 to 12
i_abc = iabc (from workspace)

Expected outputs:
faultDist = 0 to lineLength
  where lineLength is specified in the SimPowerSystems model
%

%------------------for testing purposes only----------------------
%giving the fault type a string value for generating the graphs
if(faultType == 0)
  faultTypeWord = 'NONE';
elseif(faultType == 1)
  faultTypeWord = 'AG';
elseif(faultType == 2)
  faultTypeWord = 'BG';
elseif(faultType == 3)
  faultTypeWord = 'CG';
elseif(faultType == 4)
  faultTypeWord = 'AB';
elseif(faultType == 5)
  faultTypeWord = 'BC';
elseif(faultType == 6)
  faultTypeWord = 'CA';
elseif(faultType == 7)
    faultTypeWord = 'ABG';
elseif(faultType == 8)
    faultTypeWord = 'BCG';
elseif(faultType == 9)
    faultTypeWord = 'CAG';
elseif(faultType == 10)
    faultTypeWord = 'ABCG';
elseif(faultType == 11)
    faultTypeWord = 'ABC';
end

%----for plotting the fault type that just occurred-----
faultOccWord = num2str(faultOcc);

%function
plotFaultType(I_abc,faultType,faultOcc,plotWithName,groundResistance,phaseResistance)
plotFaultType(i_abc,faultTypeWord,faultOccWord);

end
function plotFaultType(I_abc,faultType,faultOcc)

%%%%

engineer: Christina Yeoman
adviser: Dr. Yuan Liao

program purpose: The plot displays and saves the current waveform as the filename, fault type, and fault occurrence

program dependencies: none

standard inputs
I_abc = iabc (from workspace)
faultType = AG,BG,CG,AB,BC,CA,ABG,CAG,BCG,ABC,ABCG, or ERROR
faultOcc = 2 to 12

expected outputs
none

%%%%

%setting the handle for the save as function
plotFault = figure;
%plotting the current

plot(I_abc);

temp = 1 : length(I_abc(:,1));

time = (temp - 1)/(128*60);

plot(time,I_abc(:,1),time,I_abc(:,2),time,I_abc(:,3));

grid on;

xlabel('Time');

ylabel('Current (pu)');

hleg = legend('Ia','Ib','Ic',...
    'Location','NorthEastOutside');

set(hleg,'FontAngle','italic','TextColor',[.3,.2,.1]);

%plot with the title of the fault type and fault occurrence, then save as
%the desired function is with the type of fault and parameters used.

suptitle(['Fault type ' faultType ' occurred at ' faultOcc '/60']);

%saveas(plotFault, ['..\plots\plots\' faultType '_name_' faultOcc '_fO.jpg'],'jpg');

end
Appendix C

Appendix C – Simulink Model Code

function [faultType, faultOcc, estFaultDistance] = runFaultDetector(va_in,vb_in,vc_in,ia_in,ib_in,ic_in,Z0,Z1,samplesPerCycle,runTime,totalRunTime)

%#codegen

%
{
  Engineer: Christina Yeoman
  Adviser: Dr. Yuan Liao
  Program purpose: To continuously run every 15/60 seconds such that it can detect a fault and specify the fault location on the line.

  Program dependencies: faultDetectorFinal, faultDistance

  Standard inputs:
  va_in = from external input
  vb_in = from external input
  vc_in = from external input
  ia_in = from external input
  ib_in = from external input
  ic_in = from external input
  Z0 = real + imag positive sequence impedance value
  Z1 = real + imag positive sequence impedance value
  runTime = 15
  totalRunTime = 60
  faultOcc = from 1/60 to 12/60

  Expected outputs:
  faultType: 0 to 11, to external output
  faultOcc = 2/60 to 12/60, to external output
  estFaultDistance = 0 to infinity, depending on the line length and the units involved, to external output
%
}
%initializing the output parameters
faultType = 0;
faultOcc = 0;
estFaultDistance = 0;

%identifying how long to make the arrays, as well as how long to %run the while loo
iterationLength = samplesPerCycle * runTime;

%creating the arrays to detect a fault
i_abc = zeros(1921,3);
v_abc = zeros(1921,3);

%initializing the while loop parameters which will be set inside
iterations = 1;
continueWithProgram = 0;

%generating the array based on the inputs
while(iterations <= iterationLength)
    i_abc(iterations,:) = [ia_in ib_in ic_in];
    v_abc(iterations,:) = [va_in vb_in vc_in];

    if(iterations ==1281)
        continueWithProgram = 1;
    end

    iterations = iterations + 1;
end

if(continueWithProgram == 1)
    faultSet = 0;
    faultTypeTemp = 0;
    I_cycForLoc = zeros(1,3);
    V_cycForLoc = zeros(1,3);

    %calculating the run time so that we know how to iterate through
    %the time cycle properly
    runTimeFrac = runTime/totalRunTime;

    %setting the faultType array to all zeros, which means NONE
    %it will only be set to something else if there is a fault
    assert(runTime<=60);
faultTypeArr = zeros(1,runTime);
faultIforLoc = zeros(runTime,6);
faultVforLoc = zeros(runTime,6);

%this FOR loop iterates through the fault detector function and sets it to
%the fault type that has occurred, this can be NONE
%this FOR loop implements the faultDetectorFinal function
%function [faultType] =
faultDetectorFinal(I_abc,In_freq,samPerCycle,numCycleInWnd,runTime,faultOcc)
for m=1:runTime
  faultOccIter = m/totalRunTime;
  if(m < (runTime-3))
    [faultTypeOut,I_cycForLoc,V_cycForLoc] =
faultDetectorFinal(i_abc,v_abc,60,128,1,runTimeFrac,faultOccIter);
    faultTypeArr(m) = faultTypeOut;
    faultIforLoc(m,:) = I_cycForLoc;
    faultVforLoc(m,:) = V_cycForLoc;
  else
    faultTypeArr(m) = 0;
    faultIforLoc(m,:) = zeros(1,6);
    faultVforLoc(m,:) = zeros(1,6);
  end
end

%this FOR loop iterates through the faultType array which has been set to
%the fault types which have been set throughout the entire run time
%it does this by checking if the faultArray was set to anything beyond zero
for n=1:runTime
  if(faultTypeArr(n) == 0 && faultSet == 0)
    faultType = 0;
    faultOcc = 0;
  elseif(faultTypeArr(n) == 0)
    faultSet = faultSet + 1;
    if(n > 1)
      if(faultTypeArr(n) == faultTypeArr(n-1))
        faultOcc = n;
        faultType = faultTypeArr(n);
      end
    end
  end
end
if(faultType ~= 0)
    % function [estimatedDistance] =
    faultDistance(I_cycForLoc,faultTypeNum,faultOcc,groundResistance,phaseResistance)
    estFaultDistance =
    faultDistance(faultIforLoc(faultOcc,:),faultVforLoc(faultOcc,:),faultType,Z0,Z1);
    else
        estFaultDistance = 0;
    end
end

end


109


42. J. L Blackburn, Protective Relaying – Principles and Applications, Marcel Dekker, Inc. 1998

111
Christina Marie Yeoman was born in Lexington, Kentucky. She grew up in Lexington, and when she was accepted into the Computer Engineering undergraduate program at the University of Kentucky, she eagerly stayed to be close to her family and to receive an excellent education. She graduated in May of 2011, and shortly after began receiving her graduate degree in Electrical Engineering at the University of Kentucky. This thesis will be used to complete her Master’s. During her time at the university, she has been an integral part of several student and professional organizations, and has worked as an intern at several companies. Her interests are in Power Engineering, Computer Engineering, Renewable Energy, and improving the Electrical Power Grid to becoming a Smart Grid.