

GENERAL PURPOSE MICROWAVE CIRCUIT ANALYSIS USING MATLAB

By

**LEENA ARSHAD MOHAMMED AHMED
ELECTRICAL AND ELECTRONICS ENGINEERING FACULTY**

Dissertation submitted in partial fulfillment of
the requirements for the
Bachelor of Engineering (Hons.)
(Electrical & Electronics Engineering Faculty)

June 2009

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31751 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL


GENERAL PURPOSE MICROWAVE CIRCUIT ANALYSIS USING MATLAB

By

Leena Arshad Mohammed Ahmed

A project dissertation submitted to the
Electrical & Electronics Engineering Faculty
Universiti Teknologi PETRONAS
in partial fulfillment of the requirement for the
Bachelor of Engineering (Hons.)
(Electrical & Electronics Engineering)

Approved:



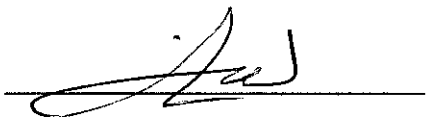
Professor Ellis, Grant Andrew
Project Supervisor

**UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK**

June 2009

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



Leena Arshad Mohammed Ahmed

ABSTRACT

The application of *Computer Aided Design* (CAD) is very crucial to microwave circuit design and analysis. The initial design of any arbitrarily microwave circuit must be simulated and verified prior to its fabrication. This eliminates the time-consuming and costly changes on the fabricated circuit due to its limited ability to incorporate any modifications. There are many commercially available CAD packages that are used in industries and universities for fabrication and academic purposes. These packages are very sophisticated and reliable. However, a licensed microwave CAD package is very expensive for universities and colleges to obtain and use, especially for academic purposes. In this final year project, an attempt to develop a computer code that acts as a basis for an alternative CAD program for microwave network design and analysis is incorporated. The CAD program is to be developed specially to suit the learning requirements and outcomes of Microwave Engineering courses at universities and colleges. This report gives a general review on microwave circuits and their representations. In addition, it demonstrates the method chosen to perform the analysis on an arbitrary connected microwave network; *Scattering Connection Matrix* method. The report also details the procedure followed to develop the required computer code based on the chosen method. The computer codes are developed using Mathematics Laboratory (Matlab) software package. A detailed discussion and verification of the results obtained is also shown in this report. The results are verified using a sophisticated microwave CAD package called *Advanced Design System* (ADS) and compared with the results obtained from implementing the developed computer codes. Comparison of both results shows an acceptable accuracy between them and thus proves that the chosen method to analyze microwave networks is very effective and reliable.

ACKNOWLEDGEMENTS

Praise be to Allah, The Most Gracious and The Most Merciful for His endless blessings throughout my life and the success He granted me during my undergraduate studies and this final year project.

My utmost appreciation and gratitude is towards my supervisor Prof. Grant A. Ellis for his guidance throughout my final year project. His knowledge, experience and support were very much helpful in passing so many obstacles that I faced. The trust he had on me pushed me forward towards achieving my goals.

My appreciation is also extended to my family members for their continuous support and sincere prayers. Special thanks to my friend Ibrahim Ali for his guidance and support throughout my project. Last but not least, I thank my friends, Sameha Ahmed, Tihani Nasser, Mojdeh Rastgoo, Tajrul Shaheer, and everyone else who encouraged and supported me throughout five years of undergraduate studies.

TABLE OF CONTENTS

ABSTRACT.....	iv
ACKNOWLEDGMENT	v
LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF ABBREVIATIONS.....	xi
CHAPTER 1 INTRODUCTION	12
1.1 Background of Study.....	12
1.2 Problem Statement.....	13
1.3 Objectives and Scope of Study.....	13
CHAPTER 2 LITERATURE REVIEW	15
2.1 Computer-Aided Design and Analysis.....	15
2.2 Microwave Circuit Representation.....	16
2.3 Connection-Scattering Matrix Method.....	18
CHAPTER 3 METHODOLOGY AND WORK FLOW	21
3.1 Overall Project Flow.....	21
3.2 Development of Project Computer Code.....	23
3.2.1 Development of Matlab Code Algorithms.....	23
3.2.2 Development of Graphical User Interface (GUI)	25
3.2.2.1 Select Network Components	26
3.2.2.2 Input Elements Parameters Values.....	26
3.2.2.3 Select Frequency of Operation and Characteristic Impedance Z_0 :.....	27
3.2.2.4 Identify Elements Ports Numbers.....	27
3.2.2.5 Calculate Scattering Parameters for the Network	27
3.2.2.6 Smith Chart Simulation	28
CHAPTER 4 RESULTS AND DISCUSSION.....	29
4.1 Results	29
4.1.1 Single Series Inductor	30
4.1.2 Single Shunt Capacitor.....	32

4.1.3 Single Microstrip Coupled-Line Filter Section.....	35
4.1.4 Single Microstrip Tee Junction.....	37
4.1.5 Single Open-End Microstrip Transmission Line	40
4.1.6 Microwave L-C Low Pass Filter Network.....	42
4.1.7 Microwave L-C High Pass Filter Network	45
4.1.8 Microwave Microstrip Coupled-Lines Band-Pass Filter	48
4.2 Discussion	52
4.2.1 Analysis of Individual Network Elements.....	52
4.2.2 Analysis of Microwave Filter Networks.....	53
CHAPTER 5 CONCLUSION AND RECOMMENDATION.....	54
5.1 Conclusion	54
5.2 Recommendations.....	55
5.2.1 Number of Network Components.....	55
5.2.2 Analysis of Other Microwave Elements.....	55
5.2.3 Simulation of Microwave Networks.....	55
REFERENCES.....	56
APPENDICES	57

LIST OF TABLES

Table 4.1: Matlab Scattering Parameters for Single Series Inductor Circuit.....	31
Table 4.2: Matlab Scattering Parameters for Single Shunt Capacitor Circuit	34
Table 4.3: Matlab Scattering Parameters for Single Microstrip Coupled-Line Filter Section Circuit	36
Table 4.4: Matlab Scattering Parameters for Single Microstrip Tee Junction Circuit	39

LIST OF FIGURES

Figure 2.1: Incident and Reflected Waves in a Two-Port Microwave Network.....	17
Figure 2.2: A Simple Multi-port, Arbitrarily Connected Microwave Network.....	18
Figure 3.1: Overall Project Procedure Flow Chart	22
Figure 3.2: Computer Code Development Procedure Flow Chart.....	24
Figure 3.3: Developed Graphical User Interface	25
Figure 4.1: Single Series Inductor Circuit	30
Figure 4.2: ADS Simulation Scattering Parameters for Single Series Inductor Circuit	30
Figure 4.3: ADS Simulation Smith Chart for Single Series Inductor Circuit.....	31
Figure 4.4: Matlab Smith Chart for Single Series Inductor Circuit.....	32
Figure 4.5: Single Shunt Capacitor Circuit.....	32
Figure 4.6: ADS Simulation Scattering Parameters for Single Shunt Capacitor Circuit	33
Figure 4.7: ADS Simulation Smith Chart for Single Shunt Capacitor Circuit.....	33
Figure 4.8: Matlab Smith Chart for Single Shunt Capacitor Circuit	34
Figure 4.9: Single Microstrip Coupled-Line Filter Section Circuit.....	35
Figure 4.10: ADS Simulation Scattering Parameters for Single Microstrip Coupled- Line Filter Section Circuit	35
Figure 4.11: ADS Simulation Smith Chart for Single Microstrip Coupled-Line Filter Section Circuit	36
Figure 4.12: Matlab Smith Chart for Single Microstrip Coupled-Line Filter Section	37
Figure 4.13: Single Microstrip Tee Junction Circuit	37
Figure 4.14: ADS Simulation Scattering Parameters for Single Microstrip Tee Junction Circuit.....	38
Figure 4.15: ADS Simulation Smith Chart for Single Microstrip Tee Junction Circuit	38
Figure 4.16: Matlab Smith Chart for Single Microstrip Tee Junction Circuit.....	39
Figure 4.17: Single Microstrip Open-End Transmission Line Circuit.....	40
Figure 4.18: ADS Simulation and Matlab Code Implementation Scattering Parameters for Single Microstrip Open-End Line Circuit.....	40

Figure 4.19: ADS Simulation Smith Chart for Single Open-End Microstrip Line Circuit	41
Figure 4.20: Matlab Smith Chart for Single Open-End Microstrip Line Circuit.....	41
Figure 4.21: Microwave L-C Low Pass Filter Network	42
Figure 4.22: ADS Simulation Scattering Parameters for Microwave L-C Low Pass Filter Network.....	43
Figure 4.23: ADS Simulation Smith Chart for Microwave L-C Low Pass Filter Network	43
Figure 4.24: Matlab Scattering Parameters for Microwave L-C Low Pass Filter Network	44
Figure 4.25: Matlab Smith Chart for Microwave L-C Low Pass Filter Network.....	44
Figure 4.26: Microwave L-C High Pass Filter Network.....	45
Figure 4.27: ADS Simulation Scattering Parameters for Microwave L-C High Pass Filter Network.....	46
Figure 4.28: ADS Simulation Smith Chart for Microwave L-C High Pass Filter Network	46
Figure 4.29: Matlab Scattering Parameters for Microwave L-C High Pass Filter Network	47
Figure 4.30: Matlab Smith Chart for Microwave L-C High Pass Filter Network	47
Figure 4.31: Microstrip Coupled-Line Band-Pass Filter Network	48
Figure 4.32: ADS Simulation Scattering Parameters for Microwave Coupled-Line Band Pass Filter Network	49
Figure 4.33: ADS Simulation Smith Chart for Microwave Coupled-Line Band Pass Filter Network.....	49
Figure 4.34: Matlab Scattering Parameters for Microwave Coupled-Line Band Pass Filter Network.....	50
Figure 4.35: Matlab Smith Chart for Microwave Coupled-Line Band Pass Filter Network	50
Figure 4.36: ADS Forward Transmission Simulation for Microwave Coupled-Line Band Pass Filter Network	51
Figure 4.37: Matlab Forward Transmission Simulation for Microwave Coupled-Line Band Pass Filter Network	51

LIST OF ABBREVIATIONS

ADS	Advanced Design System
CAE	Computer-Aided Engineering
CAD	Computer-Aided Design
GUI	Graphical User Interface
MIC	Microwave Integrated Circuit
MMIC	Monolithic Microwave Circuit

CHAPTER 1

INTRODUCTION

1.1 Background of Study

The use of Computer-Aided Design (CAD) software packages for microwave circuit design and analysis is very important and well established. There was a significant progress in Monolithic Microwave Circuit (MMIC) technologies in industry over the last decade that was highly achieved by using sophisticated microwave CAD procedures and programs ^[1].

In CAD programs for microwave network analysis, the initial design of any microwave circuit should be simulated and optimized prior to its fabrication. This eliminates the time-consuming and costly experimental investigations on the circuit after fabrication. This is of a significant importance in the design and manufacturing of modern microwave circuits because of the very limited ability to incorporate any modifications on the fabricated circuit using Monolithic Microwave Integrated Circuit (MIC) technology. Computer generated codes in CAD programs are not only used to determine the nominal values of the components parameters, but also their maximum permitted distributions in relation to a specific given condition. This is most required when a certain tolerance for circuit response function is required when a large amount of identical circuits are realized and fabricated ^[2].

1.2 Problem Statement

There are many commercially available microwave CAD packages that are effectively used in industry and at universities for fabrication and educational purposes. Most of these packages are very sophisticated and reliable but also very expensive to obtain and use specifically for academic purposes. Furthermore, these packages are closed source and specially designed to suit the fabrication requirements of microwave integrated circuits. This makes them difficult to be modified and used in an academic content.

Hence, an attempt to develop an easy to obtain and reliable open-source computer code to analyze microwave circuits is incorporated in this project. The required computer code is to be developed especially to suit the learning requirements and outcomes of Microwave Engineering courses at universities and colleges. Mathematics Laboratory 2007 (Matlab 2007) software package is chosen to develop and implement the required computer code. This software package is well known and widely used in many universities and engineering institutions. Thus, the developed computer code in this project can be easily accessed, modified and used to analyze microwave networks as required.

1.3 Objectives and Scope of Study

The purpose of this project is to develop a detailed and practical computer code that acts as a basis for a microwave network analysis and design CAD program. This alternative CAD program is to be specially used for Microwave Engineering courses at universities and engineering institutions. The required developed computer code is intended to fulfill three main objectives:

- i. To perform computerized analysis and simulation of arbitrarily connected, multi-port microwave networks with accurate and reliable results.

- ii. To be developed with a suitable Graphical User Interface (GUI) for an easy and convenient usage and implementation.
- iii. To be an open-source code that students can easily access and modify as required.

The scope of study of this project involves examining various methods and mathematical models available for analyzing microwave circuits. A detailed study of microwave circuits and their representations is also required to further understand those methods. Then based on the conducted analysis and study, the most appropriate and applicable method is chosen to be implemented in this project. A computer code that utilizes the chosen method is developed using Mathematical Laboratory 2007 (Matlab 2007) software package. The developed code is then applied to analyze and simulate some sample microwave networks. A detailed verification of the results obtained is achieved through comparison with the results obtained from Advanced Design System (ADS) software packages used for analysis and simulation of microwave networks.

CHAPTER 2

LITERATURE REVIEW

2.1 Computer-Aided Design and Analysis

Until the 1970s, RF and microwave circuit design was an art rather than a science. The common believe among people was that component modeling was inaccurate and complex. Therefore, design on the bench was the common practice in most of circuit fabrication industries at that time ^[3]. Computer-Aided Engineering (CAE) and Computer-Aided Design (CAD) for electronic circuit were born in the late 1960s and slowly gained acceptance.

CAD for microwave circuits involves repeated analysis of the circuits. The analysis consists of evaluation of the overall circuit performance parameters from the characterization of the individual components. It involves procedures used to simulate the initial circuit design and test it for accuracy and optimization prior to final circuit fabrication. RF and microwave CAD initially progressed only in the area of small-signal, linear circuit design, focusing on the analysis and optimization of discrete and hybrid microwave integrated circuits ^[3].

There are several methods and algorithms for analyzing microwave circuits that have been implemented in the recently developed CAD programs. Most of these algorithms are used to compute a certain number of response functions regarding component parameters, circuit topology and independent excitation given. Since the circuit components are usually multi-port connected, the analysis is greatly affected

by means of topological matrices that indicate the connected pairs of adjacent ports in the network [2]. Hence, any developed code for analyzing microwave circuit should take into account the connection between the adjacent components ports.

There are a number of sophisticated CAD packages that have been developed for the analysis and design of microwave circuits. Some of the early CAD packages developed were SPEEDY, which was the precursor to Compact, CAIN-01, EEsoft Touchstone which later was developed to Libra. The Engineers at Hewlett Packard had their own CAD known as MDS. They later acquired EEsoft and eventually merged Touchstone with MDS and developed a more accurate and sophisticated CAD package known as ADS [4].

2.2 Microwave Circuit Representation

In order to characterize the behavior of an arbitrary connected n -port microwave circuit, measured data of both its transfer and impedance functions must be obtained [5]. At low frequencies, the z , y , h or $ABCD$ parameters are network parameters used in the description and analysis of an arbitrarily connected n -port networks. However, these parameters cannot be measured accurately at higher frequencies (more than 1 GHz). This is because the required short-circuit and open-circuit tests are difficult to achieve over a broadband range of microwave frequencies.

A set of parameters that are applicable for the microwave range of frequencies (1GHz and above) are the *Scattering Parameters* (S-Parameters). S-Parameters are defined in terms of traveling waves (incident and reflected waves) that enter and leave the network. Incident waves are denoted as *a-waves* and the reflected waves in the network are represented as *b-waves*. These two waves represent normalized traveling voltage waves. Figure 2.1 below shows a representation of the incident and reflected waves in a two-port microwave network.

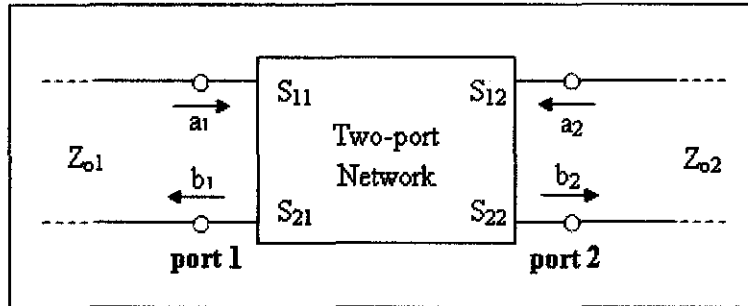


Figure 2.1: Incident and Reflected Waves in a Two-Port Microwave Network

The S-matrix equation is formed to relate the incident and reflected waves as shown below:

$$b_1 = S_{11} a_1 + S_{12} a_2 \quad \dots (2.2.1)$$

$$b_2 = S_{21} a_1 + S_{22} a_2 \quad \dots (2.2.2)$$

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad \dots (2.2.3)$$

In general, S-matrices are used for the characterization of microwave circuits. Hence, two-port networks can be combined arbitrarily in series or parallel to yield multi-port (n-port) microwave network. These networks can then be analyzed by using any of the multi-port connection methods.

There are two main methods used to analyze arbitrarily connected n-port microwave networks:

1. Analysis Using Connection-Scattering Matrix.
2. Multi-port Connection Method.

Regardless of the method used in the analysis, the S-matrix of the multi-port connection is required.

2.3 Connection-Scattering Matrix Method

This method is chosen to develop the computer code for microwave network analysis in this project. It is applicable when the network contains arbitrarily interconnected ports of several elements and independent external input and output ports (*m components*). A block diagram of such a network is shown Figure 2.2 below. The arrows in the network represent the directions of the incident and reflected waves at each element port or node on the network. The analysis involves evaluation of the scattering parameters of individual elements of the circuit in connection with the information on circuit topology and setting them up in the form of the connection scattering matrix. Microwave circuits with any arbitrary topology may be analyzed using this matrix formalism ^[6].

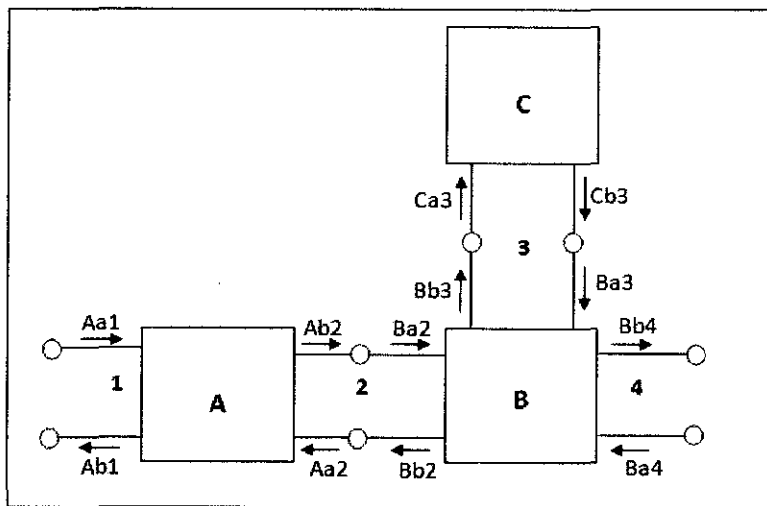


Figure 2.2: A Simple Multi-port, Arbitrarily Connected Microwave Network

To illustrate this method clearly, consider that the governing relations for all the *m* components in the network can be put together in the form:

$$\mathbf{b} = \mathbf{S} \mathbf{a} + \mathbf{c} \quad \dots (2.3.1)$$

$$a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}, \quad c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

where a : Incident waves vector
 b : Reflected waves vector
 c : Network excitation vector

and

$$S = \begin{bmatrix} S_1 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & S_i & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & S_m \end{bmatrix} \quad \dots (2.3.2)$$

The matrix in (2.3.2) is called *Scattering Matrix* and it represents a block diagonal matrix whose sub-matrices along the diagonal are the scattering matrices of various m components of the network and 0s represent null matrices.

In Figure 2, it is clearly shown that for a pair of connected ports, the outgoing wave variable at one port must be equal to the incoming wave variable at the other port. Assuming connected ports j and k , the incoming and outgoing waves satisfy:

$$a_j = b_k \quad \dots (2.3.3)$$

$$a_k = b_j \quad \dots (2.3.4)$$

or

$$\begin{bmatrix} b_k \\ b_j \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a_k \\ a_j \end{bmatrix} \quad \dots (2.3.5)$$

The matrix in (2.3.5) is known as the *inverse of the S-matrix of the interconnection*. This relation can be written for all the interconnected ports in the network in the form:

$$\mathbf{b} = \mathbf{\Gamma} \mathbf{a} \quad \dots (2.3.6)$$

where $\mathbf{\Gamma}$ is a *Connection Matrix* describing the topology of the interconnected network.

Hence, from (2.3.1) and (2.3.6):

$$\mathbf{\Gamma} \mathbf{a} = \mathbf{S} \mathbf{a} + \mathbf{c}$$

or $(\mathbf{\Gamma} - \mathbf{S}) \mathbf{a} = \mathbf{c}$

Setting $\mathbf{W} = \mathbf{\Gamma} - \mathbf{S}$

$$\mathbf{a} = \mathbf{W}^{-1} \mathbf{c} \quad \dots (2.3.7)$$

where \mathbf{W} is called the *Connection-Scattering Matrix*.

The analysis of an n-port arbitrarily connected microwave network is determined from equations (2.3.6) and (2.3.7). The solution of (2.3.7) gives the incoming waves \mathbf{a} at all the components ports in the network. Then the outgoing waves \mathbf{b} can be obtained from (2.3.6). According to (2.2.3), both \mathbf{a} waves and \mathbf{b} waves are then used to determine the overall S parameters for the overall network.

CHAPTER 3

METHODOLOGY AND WORK FLOW

3.1 Overall Project Flow

In order to develop the required computer code for microwave circuit analysis and simulation, a set of procedures were followed:

- 1. Identify problem, Objectives and Scope:** Conduct a research to examine and understand various available methods and mathematical models for microwave circuit analysis based on the problem statement, objectives and scope of the project.
- 2. Selecting Feasible and Reliable Method:** Based on the conducted research, the most feasible, applicable and reliable method which is the Connection-Scattering Matrix method is chosen.
- 3. Develop Matlab Code:** Based on the Connection-Scattering Matrix method chosen, the computer code algorithms required are developed using Matlab 2007 software package.
- 4. Test the Developed Code:** The developed code is implemented on sample microwave networks and results obtained are verified to check whether it meets the project requirements or not.
- 5. Apply Modifications:** The developed code is then examined for further improvement and modifications.

Figure 3.1 below shows a Flow Chart of the overall procedure chosen to implement the desired computer code.

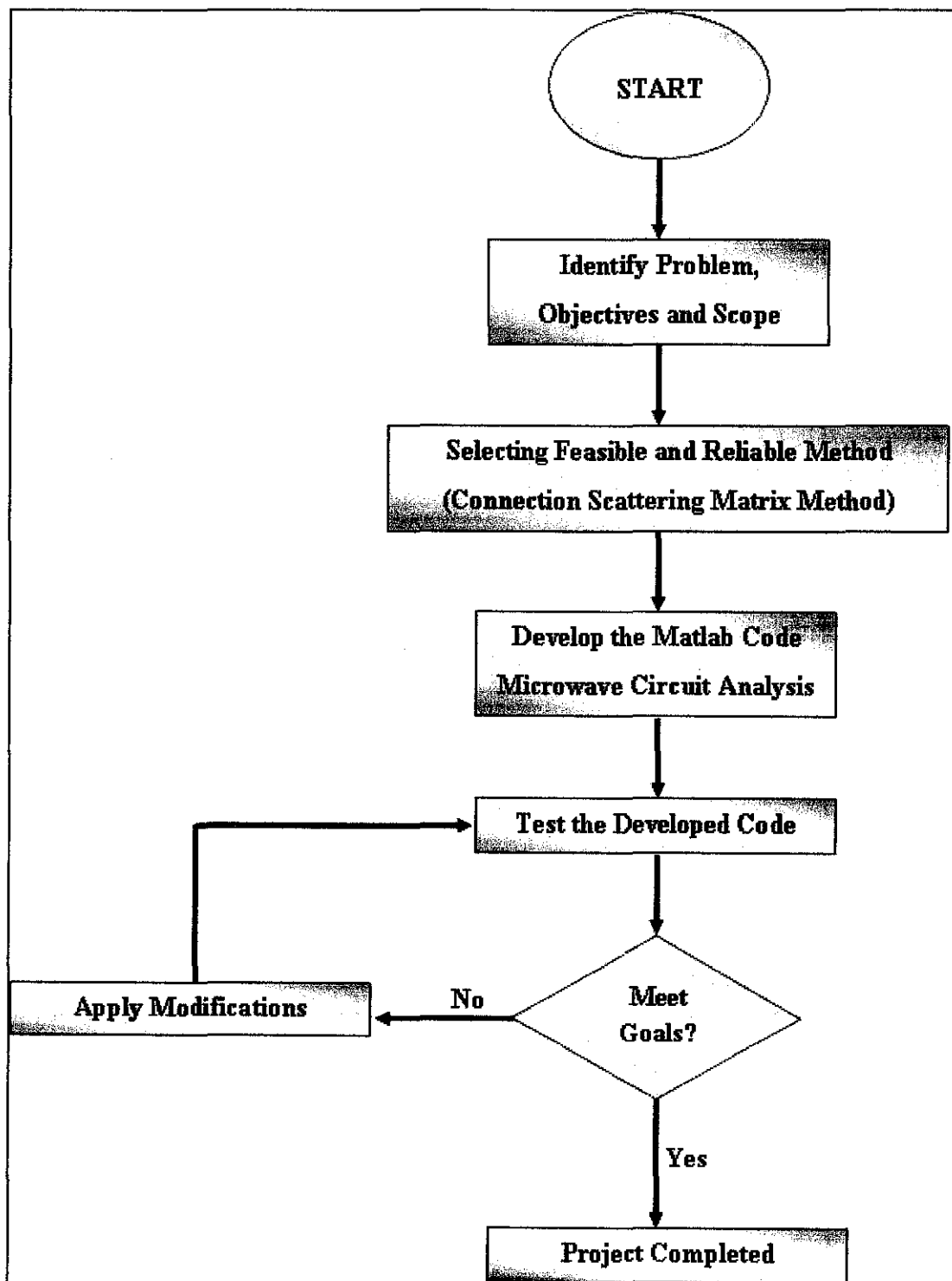


Figure 3.1: Overall Project Procedure Flow Chart

3.2 Development of Project Computer Code

The required computer code in this final year project is chosen to be developed using Mathematical Laboratory 2007 (Matlab 2007) software package. Matlab is specifically chosen because it is widely known and used extensively in universities and colleges to apply and verify mathematical theories in practical forms. It also incorporates many useful features such as there is no need for declaration of variables, simple and convenient syntax, easy creation of Graphical User Interfaces (GUIs) and incorporating many simulation and visualization features.

3.2.1 Development of Matlab Code Algorithms

The Connection-Scattering Matrix method is chosen to develop the computer code algorithms. As it has been mentioned earlier, the analysis of an arbitrarily connected multi-port microwave network based on the Connection-Scattering Matrix method depends on determining two major matrices, Scattering Matrix and Connection Matrix. From both these matrices, Connection-Scattering Matrix is then formed and used to determine the scattering parameters for a given microwave network.

The first step in order to develop the required computer code is to write individual Matlab functions that are used to determine the scattering parameters for several microwave elements based on their user defined values and descriptions. Another Matlab function is then developed to put the scattering parameters of each individual element in any given microwave network in one matrix called the Scattering Matrix. Information on the network's topology and how elements are interconnected with each other are used to generate the Connection Matrix using several other Matlab functions. After that, some Matlab functions are developed to take both matrices and determine the Scattering Parameters of the overall network based on certain default network excitations normally applied in microwave network analysis.

Figure 3.2 below is a flow chart that represents the steps taken to develop the required computer code for microwave circuit analysis in this project. All the developed Matlab codes are shown in Appendix A.

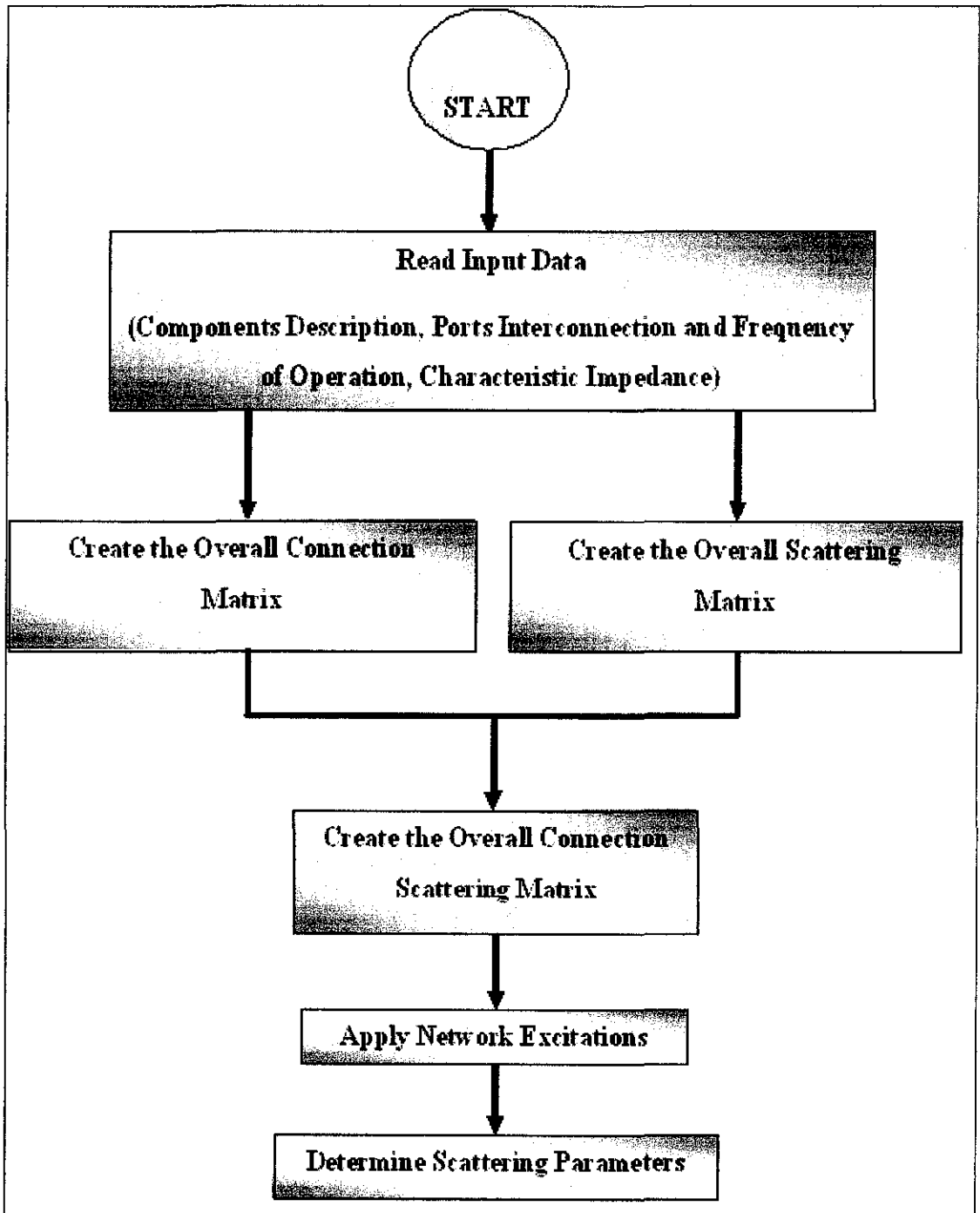


Figure 3.2: Computer Code Development Procedure Flow Chart

3.2.2 Development of Graphical User Interface (GUI)

A simple and suitable Graphical User Interface (GUI) is developed using Matlab 2007 software package to facilitate the user to implement the developed computer code for microwave circuit analysis. The developed GUI for this purpose is named Microwave Circuit Analysis Program and is shown in Figure 3.3 below. It can be easily activated by typing the command line 'microwave_gui' on Matlab workspace screen.

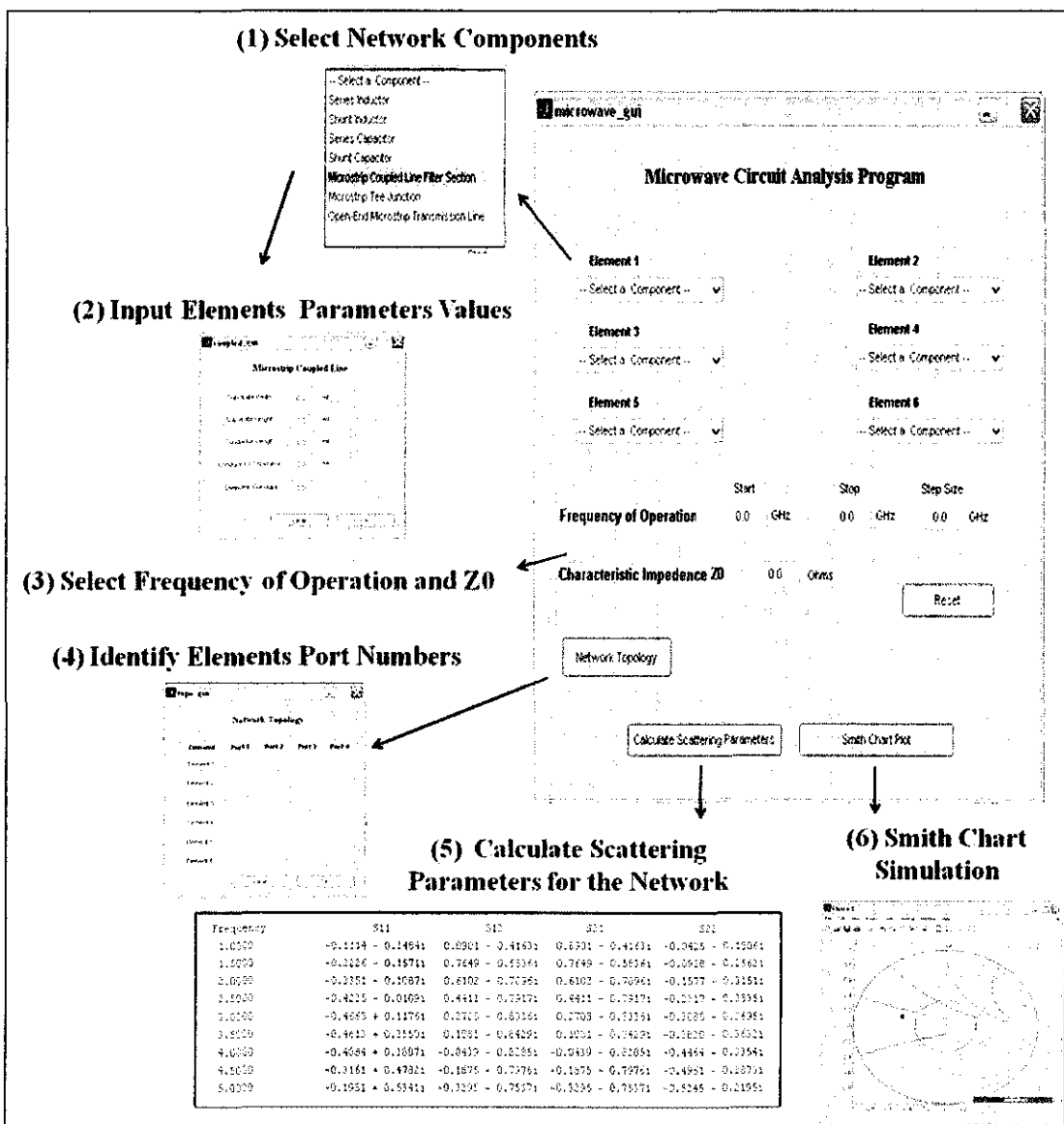


Figure 3.3: Developed Graphical User Interface

Through this Program, the user can enter all input data needed to calculate both the Scattering and Connection Matrices required to determine the Scattering Parameters for a given microwave network. In addition, the program enables the user to simulate the given network by plotting the Smith Chart for the calculated reflection coefficient vector (S11 vector). Figure 3.3 shows the steps the user follows when using the Microwave Circuit Analysis Program to analyze and simulate any arbitrarily connected microwave network as follows:

3.2.2.1 Select Network Components

The program allows the user to select up to six different network components. For each component, the user can select an element from a menu that contains different elements normally used in microwave network design. For the purpose of this final year project, the user selects an element from a menu that contains seven options; which are:

- Series Inductor
- Shunt Inductor
- Series Capacitor
- Shunt Capacitor
- Microstrip Coupled-Line Filter Section
- Microstrip Tee Junction
- Open-End Microstrip Transmission Line

3.2.2.2 Input Elements Parameters Values

Once the user selects an element from the menu, a pop up window appears through which the user can input the element's parameters values. The user can clear those values and retype them back any time by clicking on 'Clear'. Once the values are entered, the user clicks on 'OK' and continue with the following element in the network.

After the user enters all the parameters values for all elements in the network, the program creates sets of data files that contain elements parameters input values.

These files can be retrieved at anytime and data can be read and used to analyze the network as required.

3.2.2.3 Select Frequency of Operation and Characteristic Impedance Z_0 :

The user selects a range of frequencies at which the given network is analyzed. The user determines the start and stop frequencies values of the given frequency range as well as the step size of the frequency increment in the range. The developed program allows the user to enter frequency value in the GHz range. In addition, the user has to specify the characteristic impedance (Z_0) at which the analysis is performed.

From the data entered previously, the program takes these values and uses them to create the Scattering Matrix for all selected elements in the network for the range of frequencies given.

3.2.2.4 Identify Elements Ports Numbers

The user is also required to provide information on the network topology. When the user clicks on 'Network Topology', a pop up window appears that enables the user to enter the elements ports numbers. Four ports places allocated for each network element based on its type. When the user clicks on 'OK', the program will take the elements ports numbers given by the user and use them to generate the Connection Matrix for the network.

3.2.2.5 Calculate Scattering Parameters for the Network

When the user clicks on 'Calculate Scattering Parameters', the program takes both Scattering and Connection Matrices and determines the Scattering Parameters for the overall network for the range of frequencies specified by the user in the program. The Scattering Parameters appears on Matlab workspace screen in a form of a table with each row in the table representing the scattering parameters for the network for one frequency at a time.

3.2.2.6 *Smith Chart Simulation*

The developed program incorporates the feature of simulating the response of the given network through plotting the Smith Chart simulation of the calculated reflection coefficients, S11 values. When the user clicks on 'Smith Chart Plot', the program takes the S11 values from the first column from the results table as a vector and converts them into load impedances vector. The program then uses this load impedances vector to plot the Smith Chart simulation for the network. The Matlab code used to plot the Smith Chart is originally developed by Antony-Dean McKechnie & Neville Wilken in their final year project at Wits, South Africa. It was then further developed by Alan Robert Clark, Department of Electrical Engineering, Wits, South Africa, 1992. For the purpose of this final year project, this Smith Chart Matlab code was implemented with minor modifications to suit the objective of the project. The Smith Chart and the GUIs Matlab functions are shown in Appendix A.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Results

For the purpose of this final year project, the required computer code was developed using Matlab 2007 software package as indicated in Figure 3.2. As explained earlier, several Matlab functions were first developed to determine the Scattering Parameters for several microwave elements. Those codes were implemented to analyze and simulate each element individually. The results obtained are compared with the results obtained from analyzing those elements using Advanced Design System (ADS) software package used for design and analysis of microwave networks. The analysis of these elements and the verification of the obtained results are shown in the following sections of this chapter.

Once the developed codes for the individual elements were verified, the Graphical User Interface, Microwave Circuit Analysis Program, was developed to analyze and simulate microwave networks as shown in Figure 3.3. The program takes input data from the user and creates both Scattering and Connection Matrices then uses them to calculate the Scattering Parameters for the overall network. The program was used to analyze and simulate three sample microwave networks, an L-C Low Pass Filter, an L-C High Pass Filter, and Microstrip Coupled-Line Band Pass Filter. The results obtained are also compared with the results obtained from analyzing those networks using Advanced Design System (ADS) software package. The analysis of these networks and the verification of the obtained results are also shown in the following sections of this chapter.

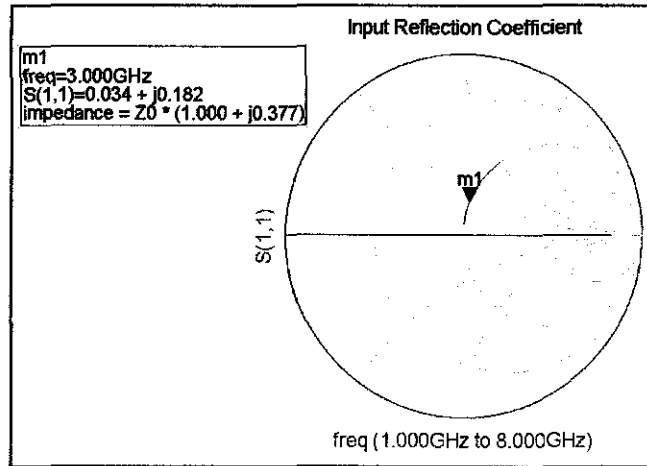


Figure 4.3: ADS Simulation Smith Chart for Single Series Inductor Circuit

Table 4.1: Matlab Scattering Parameters for Single Series Inductor Circuit

Freq (GHz)	S(1,1)	S(1,2)	S(2,1)	S(2,2)
1.0	0.0039 + 0.0626i	0.9961 - 0.0626i	0.9961 - 0.0626i	0.0039 + 0.0626i
1.5	0.0088 + 0.0934i	0.9912 - 0.0934i	0.9912 - 0.0934i	0.0088 + 0.0934i
2.0	0.0155 + 0.1237i	0.9845 - 0.1237i	0.9845 - 0.1237i	0.0155 + 0.1237i
2.5	0.0241 + 0.1533i	0.9759 - 0.1533i	0.9759 - 0.1533i	0.0241 + 0.1533i
3.0	0.0343 + 0.1820i	0.9657 - 0.1820i	0.9657 - 0.1820i	0.0343 + 0.1820i
3.5	0.0461 + 0.2098i	0.9539 - 0.2098i	0.9539 - 0.2098i	0.0461 + 0.2098i
4.0	0.0594 + 0.2364i	0.9406 - 0.2364i	0.9406 - 0.2364i	0.0594 + 0.2364i
4.5	0.0740 + 0.2618i	0.9260 - 0.2618i	0.9260 - 0.2618i	0.0740 + 0.2618i
5.0	0.0898 + 0.2859i	0.9102 - 0.2859i	0.9102 - 0.2859i	0.0898 + 0.2859i



Figure 4.3: ADS Simulation Smith Chart for Single Series Inductor Circuit

Table 4.1: Matlab Scattering Parameters for Single Series Inductor Circuit

Freq (GHz)	S(1,1)	S(1,2)	S(2,1)	S(2,2)
1.0	0.0039 + 0.0626i	0.9961 - 0.0626i	0.9961 - 0.0626i	0.0039 + 0.0626i
1.5	0.0088 + 0.0934i	0.9912 - 0.0934i	0.9912 - 0.0934i	0.0088 + 0.0934i
2.0	0.0155 + 0.1237i	0.9845 - 0.1237i	0.9845 - 0.1237i	0.0155 + 0.1237i
2.5	0.0241 + 0.1533i	0.9759 - 0.1533i	0.9759 - 0.1533i	0.0241 + 0.1533i
3.0	0.0343 + 0.1820i	0.9657 - 0.1820i	0.9657 - 0.1820i	0.0343 + 0.1820i
3.5	0.0461 + 0.2098i	0.9539 - 0.2098i	0.9539 - 0.2098i	0.0461 + 0.2098i
4.0	0.0594 + 0.2364i	0.9406 - 0.2364i	0.9406 - 0.2364i	0.0594 + 0.2364i
4.5	0.0740 + 0.2618i	0.9260 - 0.2618i	0.9260 - 0.2618i	0.0740 + 0.2618i
5.0	0.0898 + 0.2859i	0.9102 - 0.2859i	0.9102 - 0.2859i	0.0898 + 0.2859i

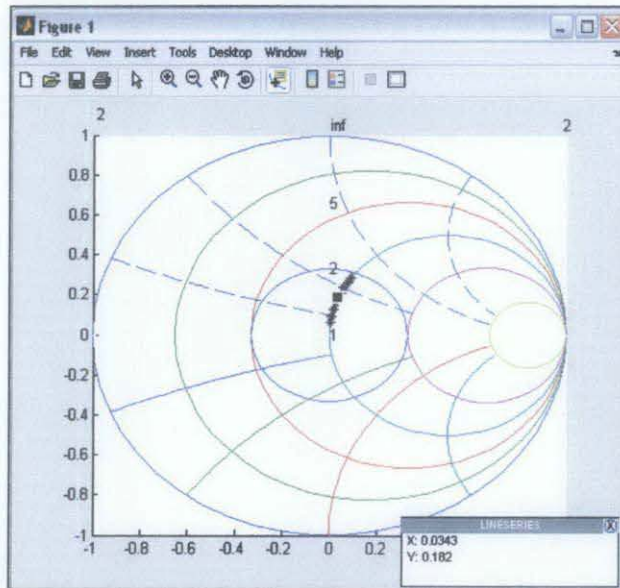


Figure 4.4: Matlab Smith Chart for Single Series Inductor Circuit

4.1.2 Single Shunt Capacitor

A single shunt capacitor circuit is shown in Figure 4.5 below. The analysis of the element is performed at characteristic impedance of 50 Ohms and specified range of frequencies of (1.0 -5.0) GHz with 0.5 GHz increment. Results of both ADS simulation and Matlab code implementation are shown clearly. Smith Chart simulation results are also clearly indicated. A value pointer is placed at 3GHz frequency point for each plot to verify the results obtained.

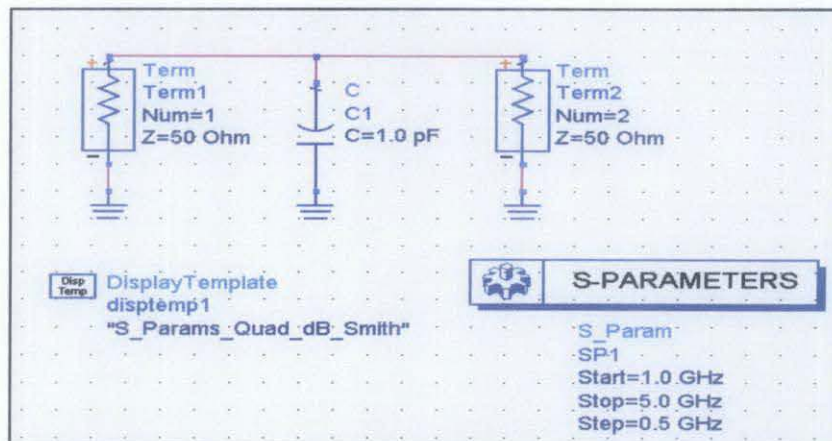


Figure 4.5: Single Shunt Capacitor Circuit

freq	S(1,1)	S(1,2)	S(2,1)	S(2,2)
1.000 GHz	-0.024 - j0.153	0.976 - j0.153	0.976 - j0.153	-0.024 - j0.153
1.500 GHz	-0.053 - j0.223	0.947 - j0.223	0.947 - j0.223	-0.053 - j0.223
2.000 GHz	-0.090 - j0.286	0.910 - j0.286	0.910 - j0.286	-0.090 - j0.286
2.500 GHz	-0.134 - j0.340	0.866 - j0.340	0.866 - j0.340	-0.134 - j0.340
3.000 GHz	-0.182 - j0.386	0.818 - j0.386	0.818 - j0.386	-0.182 - j0.386
3.500 GHz	-0.232 - j0.422	0.768 - j0.422	0.768 - j0.422	-0.232 - j0.422
4.000 GHz	-0.283 - j0.450	0.717 - j0.450	0.717 - j0.450	-0.283 - j0.450
4.500 GHz	-0.333 - j0.471	0.667 - j0.471	0.667 - j0.471	-0.333 - j0.471
5.000 GHz	-0.382 - j0.486	0.618 - j0.486	0.618 - j0.486	-0.382 - j0.486

Figure 4.6: ADS Simulation Scattering Parameters for Single Shunt Capacitor Circuit

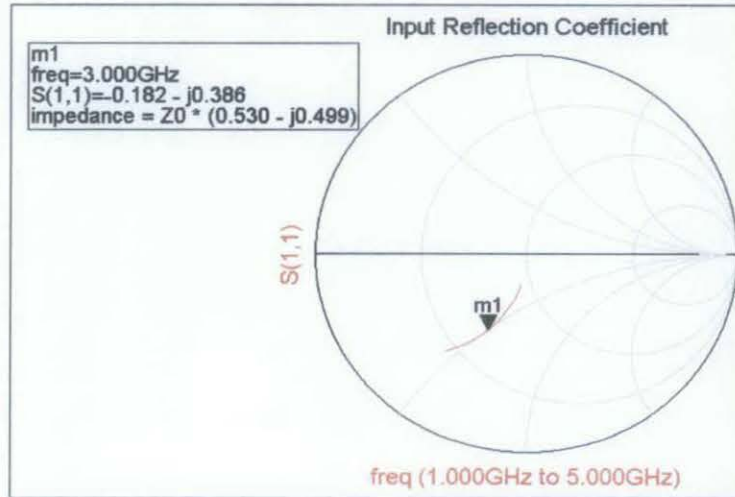


Figure 4.7: ADS Simulation Smith Chart for Single Shunt Capacitor Circuit

Table 4.2: Matlab Scattering Parameters for Single Shunt Capacitor Circuit

Freq (GHz)	S(1,1)	S(1,2)	S(2,1)	S(2,2)
1.0	-0.0241 - 0.1533i	0.9759 - 0.1533i	0.9759 - 0.1533i	-0.0241 - 0.1533i
1.5	-0.0526 - 0.2232i	0.9474 - 0.2232i	0.9474 - 0.2232i	-0.0526 - 0.2232i
2.0	-0.0898 - 0.2859i	0.9102 - 0.2859i	0.9102 - 0.2859i	-0.0898 - 0.2859i
2.5	-0.1336 - 0.3402i	0.8664 - 0.3402i	0.8664 - 0.3402i	-0.1336 - 0.3402i
3.0	-0.1817 - 0.3856i	0.8183 - 0.3856i	0.8183 - 0.3856i	-0.1817 - 0.3856i
3.5	-0.2321 - 0.4222i	0.7679 - 0.4222i	0.7679 - 0.4222i	-0.2321 - 0.4222i
4.0	-0.2830 - 0.4505i	0.7170 - 0.4505i	0.7170 - 0.4505i	-0.2830 - 0.4505i
4.5	-0.3332 - 0.4713i	0.6668 - 0.4713i	0.6668 - 0.4713i	-0.3332 - 0.4713i
5.0	-0.3815 - 0.4858i	0.6185 - 0.4858i	0.6185 - 0.4858i	-0.3815 - 0.4858i

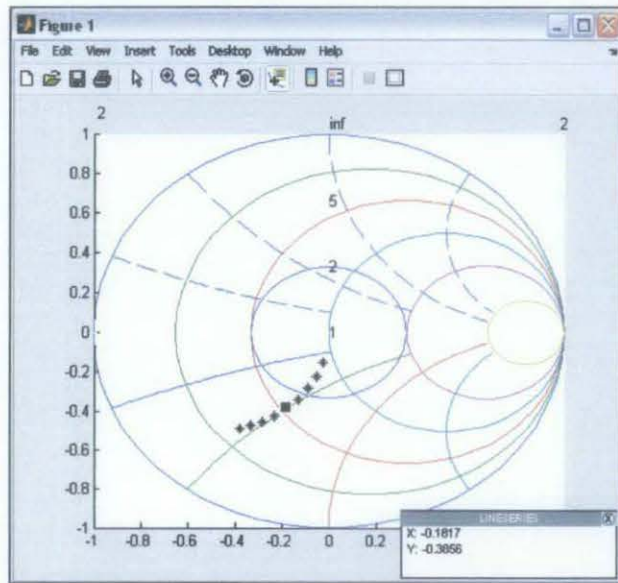


Figure 4.8: Matlab Smith Chart for Single Shunt Capacitor Circuit

4.1.3 Single Microstrip Coupled-Line Filter Section

A single microstrip coupled-line filter section circuit is shown in Figure 4.9 below. The analysis of the element is performed at characteristic impedance of 50 Ohms and specified range of frequencies of (1.0 -5.0) GHz with 0.5 GHz increment. Results of both ADS simulation and Matlab code implementation are shown clearly. Smith Chart simulation results are also clearly indicated. A value pointer is placed at 3GHz frequency point for each plot to verify the results obtained.

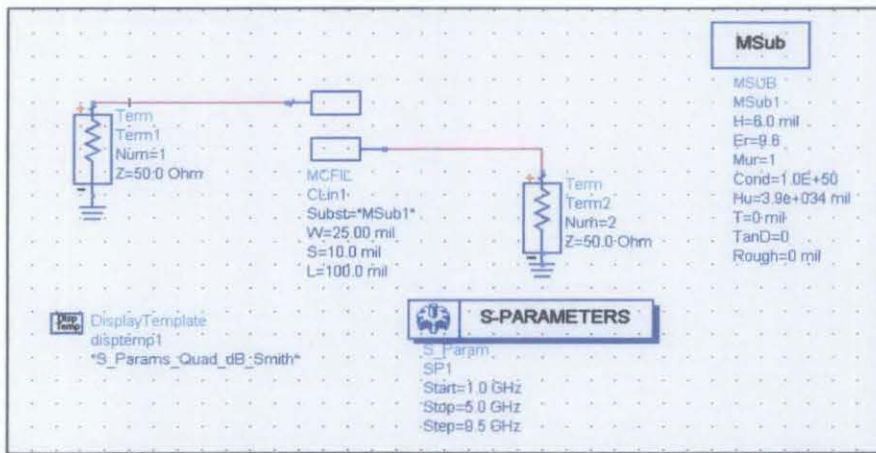


Figure 4.9: Single Microstrip Coupled-Line Filter Section Circuit

freq	S(1,1)	S(1,2)	S(2,1)	S(2,2)
1.000 GHz	0.762 - j0.647	0.007 + j0.008	0.007 + j0.008	0.762 - j0.647
1.500 GHz	0.528 - j0.849	0.013 + j0.008	0.013 + j0.008	0.528 - j0.849
2.000 GHz	0.278 - j0.960	0.017 + j0.005	0.017 + j0.005	0.278 - j0.960
2.500 GHz	0.044 - j0.999	0.020 + j8.684E-4	0.020 + j8.684E-4	0.044 - j0.999
3.000 GHz	-0.159 - j0.987	0.021 - j0.003	0.021 - j0.003	-0.159 - j0.987
3.500 GHz	-0.330 - j0.944	0.021 - j0.007	0.021 - j0.007	-0.330 - j0.944
4.000 GHz	-0.469 - j0.883	0.020 - j0.011	0.020 - j0.011	-0.469 - j0.883
4.500 GHz	-0.583 - j0.812	0.019 - j0.014	0.019 - j0.014	-0.583 - j0.812
5.000 GHz	-0.674 - j0.738	0.018 - j0.017	0.018 - j0.017	-0.674 - j0.738

Figure 4.10: ADS Simulation Scattering Parameters for Single Microstrip Coupled-Line Filter Section Circuit

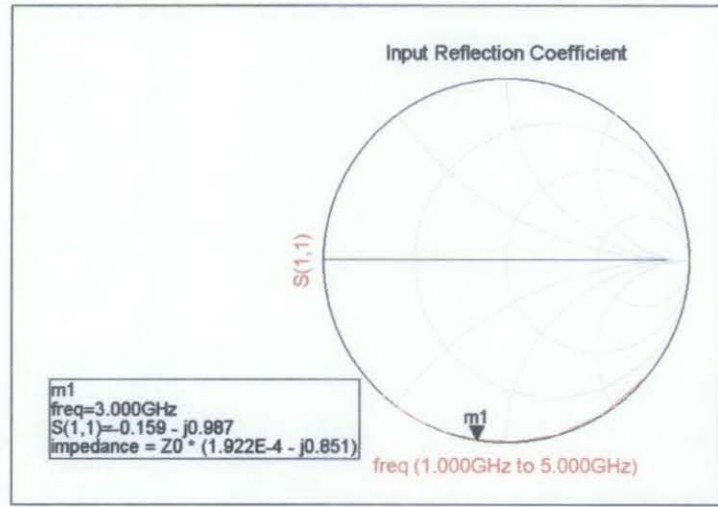


Figure 4.11: ADS Simulation Smith Chart for Single Microstrip Coupled-Line Filter Section Circuit

Table 4.3: Matlab Scattering Parameters for Single Microstrip Coupled-Line Filter Section Circuit

Freq (GHz)	S(1,1)	S(1,2)	S(2,1)	S(2,2)
1.0	0.7706 - 0.6364i	0.0213 + 0.0257i	0.0213 + 0.0257i	0.7706 - 0.6364i
1.5	0.5427 - 0.8387i	0.0375 + 0.0242i	0.0375 + 0.0242i	0.5427 - 0.8387i
2.0	0.2988 - 0.9529i	0.0493 + 0.0155i	0.0493 + 0.0155i	0.2988 - 0.9529i
2.5	0.0688 - 0.9961i	0.0552 + 0.0038i	0.0552 + 0.0038i	0.0688 - 0.9961i
3.0	-0.1330 - 0.9895i	0.0562 - 0.0076i	0.0562 - 0.0076i	-0.1330 - 0.9895i
3.5	-0.3032 - 0.9512i	0.0539 - 0.0172i	0.0539 - 0.0172i	-0.3032 - 0.9512i
4.0	-0.4437 - 0.8944i	0.0498 - 0.0247i	0.0498 - 0.0247i	-0.4437 - 0.8944i
4.5	-0.5585 - 0.8278i	0.0449 - 0.0303i	0.0449 - 0.0303i	-0.5585 - 0.8278i
5.0	-0.6518 - 0.7566i	0.0399 - 0.0344i	0.0399 - 0.0344i	-0.6518 - 0.7566i

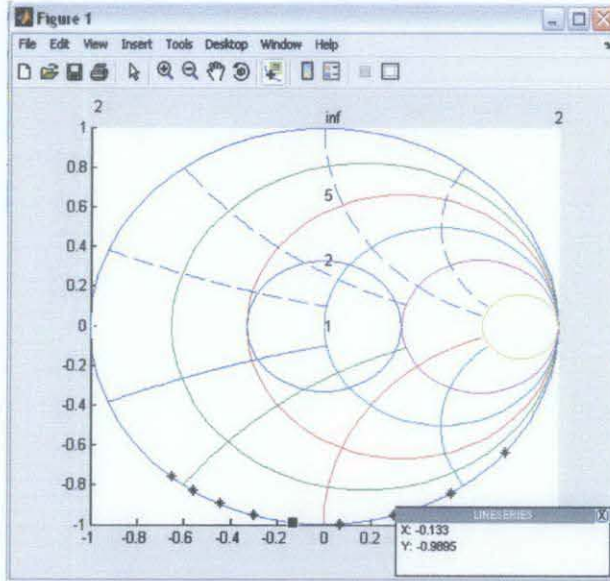


Figure 4.12: Matlab Smith Chart for Single Microstrip Coupled-Line Filter Section

4.1.4 Single Microstrip Tee Junction

A single microstrip tee junction circuit is shown in Figure 4.13 below. The analysis of the element is performed at characteristic impedance of 50 Ohms and specified range of frequencies of (15.0 -20.0) GHz with 0.5 GHz increment. Results of both ADS simulation and Matlab code implementation are shown clearly. Smith Chart simulation results are also clearly indicated. A value pointer is placed at 17GHz frequency point for each plot to verify the results obtained.

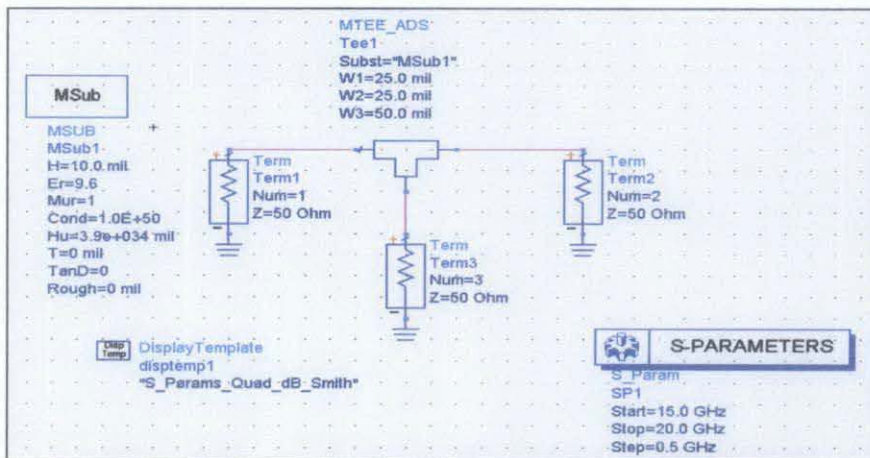


Figure 4.13: Single Microstrip Tee Junction Circuit

freq	S(1,1)	S(1,2)	S(1,3)	S(3,3)
15.00 GHz	-0.409 + j0.007	0.450 - j0.504	0.529 - j0.309	-0.453 - j0.209
15.50 GHz	-0.416 + j0.011	0.430 - j0.522	0.516 - j0.321	-0.465 - j0.214
16.00 GHz	-0.424 + j0.015	0.409 - j0.539	0.502 - j0.333	-0.477 - j0.218
16.50 GHz	-0.431 + j0.020	0.386 - j0.556	0.487 - j0.344	-0.490 - j0.222
17.00 GHz	-0.439 + j0.026	0.363 - j0.572	0.472 - j0.355	-0.503 - j0.224
17.50 GHz	-0.446 + j0.033	0.339 - j0.587	0.456 - j0.365	-0.517 - j0.225
18.00 GHz	-0.453 + j0.041	0.313 - j0.602	0.439 - j0.375	-0.531 - j0.226
18.50 GHz	-0.459 + j0.051	0.287 - j0.616	0.422 - j0.384	-0.546 - j0.225
19.00 GHz	-0.464 + j0.061	0.260 - j0.628	0.405 - j0.392	-0.561 - j0.223
19.50 GHz	-0.469 + j0.072	0.232 - j0.641	0.388 - j0.400	-0.576 - j0.220
20.00 GHz	-0.473 + j0.084	0.203 - j0.652	0.370 - j0.407	-0.590 - j0.215

Figure 4.14: ADS Simulation Scattering Parameters for Single Microstrip Tee Junction Circuit

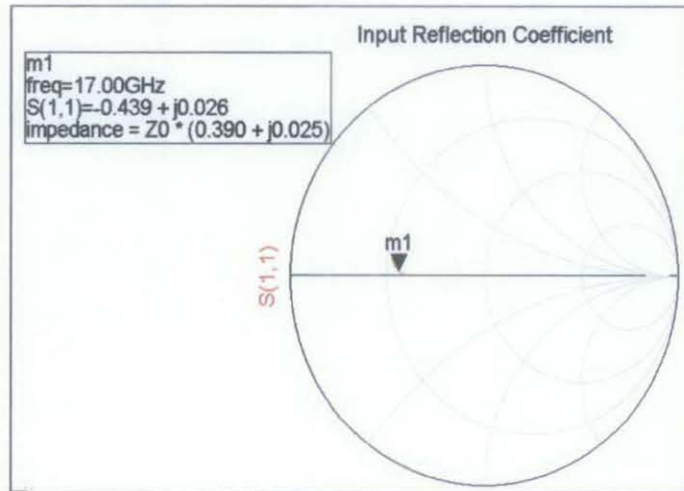


Figure 4.15: ADS Simulation Smith Chart for Single Microstrip Tee Junction Circuit

Table 4.4: Matlab Scattering Parameters for Single Microstrip Tee Junction Circuit

Freq (GHz)	S(1,1)	S(1,2)	S(1,3)	S(3,3)
15.0	-0.3280 - 0.0733i	0.6718 + 0.0017i	0.6562 + 0.0716i	-0.3315 - 0.0496i
15.5	-0.3276 - 0.0757i	0.6721 + 0.0019i	0.6555 + 0.0739i	-0.3314 - 0.0512i
16.0	-0.3273 - 0.0781i	0.6725 + 0.0020i	0.6548 + 0.0761i	-0.3312 - 0.0529i
16.5	-0.3269 - 0.0805i	0.6728 + 0.0022i	0.6541 + 0.0784i	-0.3311 - 0.0545i
17.0	-0.3265 - 0.0829i	0.6732 + 0.0023i	0.6533 + 0.0806i	-0.3310 - 0.0562i
17.5	-0.3261 - 0.0853i	0.6736 + 0.0025i	0.6525 + 0.0828i	-0.3308 - 0.0578i
18.0	-0.3257 - 0.0877i	0.6739 + 0.0027i	0.6518 + 0.0850i	-0.3307 - 0.0595i
18.5	-0.3253 - 0.0901i	0.6743 + 0.0028i	0.6510 + 0.0872i	-0.3305 - 0.0611i
19.0	-0.3249 - 0.0924i	0.6747 + 0.0030i	0.6501 + 0.0894i	-0.3304 - 0.0628i

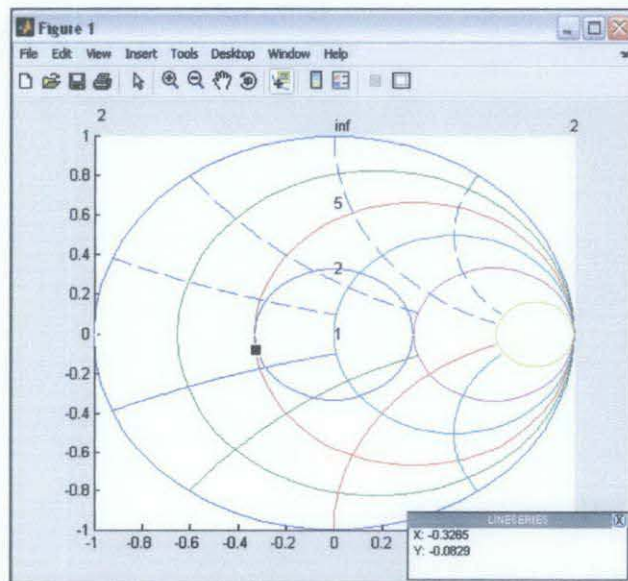


Figure 4.16: Matlab Smith Chart for Single Microstrip Tee Junction Circuit

4.1.5 Single Open-End Microstrip Transmission Line

A single open-end microstrip transmission line circuit is shown in Figure 4.17 below. The analysis of the element is performed at characteristic impedance of 50 Ohms and specified range of frequencies of (1.0 -5.0) GHz with 0.5 GHz increment. Results of both ADS simulation and Matlab code implementation are shown clearly. Smith Chart simulation results are also clearly indicated. A value pointer is placed at 3GHz frequency point for each plot to verify the results obtained.

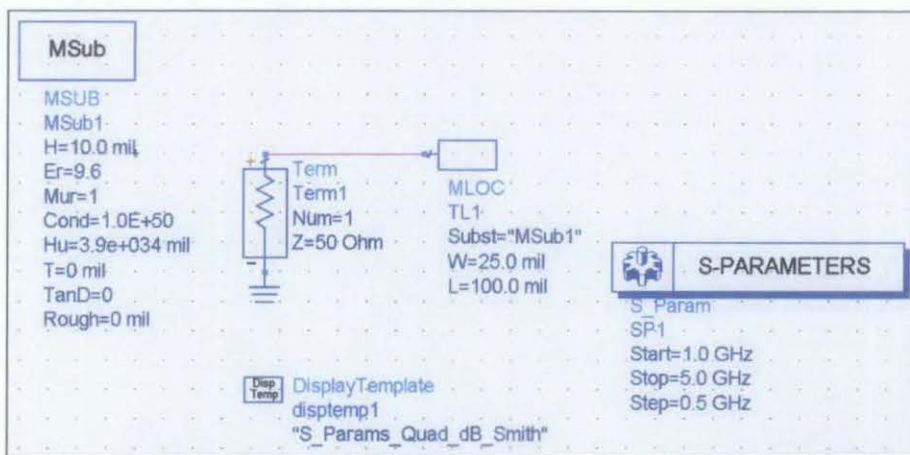


Figure 4.17: Single Microstrip Open-End Transmission Line Circuit

ADS Simulation		Matlab Code Implementation	
Freq (GHz)	S(1,1)	freq	S(1,1)
1.0	0.8814 - 0.4724i	1.000 GHz	0.889 - j0.458
1.5	0.7475 - 0.6643i	1.500 GHz	0.763 - j0.647
2.0	0.5828 - 0.8126i	2.000 GHz	0.606 - j0.796
2.5	0.4024 - 0.9155i	2.500 GHz	0.432 - j0.902
3.0	0.2192 - 0.9757i	3.000 GHz	0.254 - j0.967
3.5	0.0426 - 0.9991i	3.500 GHz	0.081 - j0.997
4.0	-0.1216 - 0.9926i	4.000 GHz	-0.082 - j0.997
4.5	-0.2702 - 0.9628i	4.500 GHz	-0.231 - j0.973
5.0	-0.4022 - 0.9156i	5.000 GHz	-0.365 - j0.931

Figure 4.18: ADS Simulation and Matlab Code Implementation Scattering Parameters for Single Microstrip Open-End Line Circuit

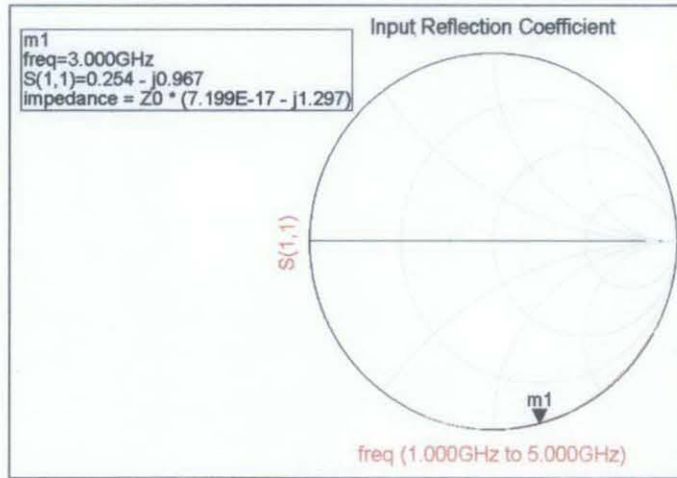


Figure 4.19: ADS Simulation Smith Chart for Single Open-End Microstrip Line Circuit

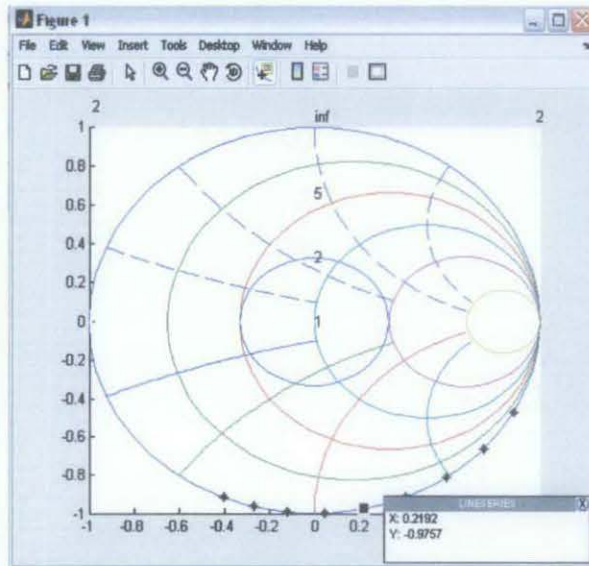


Figure 4.20: Matlab Smith Chart for Single Open-End Microstrip Line Circuit

4.1.6 Microwave L-C Low Pass Filter Network

Figure 4.21 represents a microwave L-C Low Pass Filter network. This filter network is considered to be a four cascaded two-port systems with the inductors and capacitors being the four cascaded systems respectively. The characteristic impedance of the network is set to 50 Ohms and the frequency of operation is within the range of 1.0 – 5.0 GHz with an increment of 0.5 GHz.

The developed Matlab program was used to enter all input data of the network's elements parameters values and ports numbers and analyze the network at the specified range of frequencies. The same network was then analyzed using ADS software package. Results of both ADS simulation and Matlab code implementation are shown clearly. Smith Chart simulation results are also clearly indicated. A value pointer is placed at 3GHz frequency point for each plot to verify the results obtained.

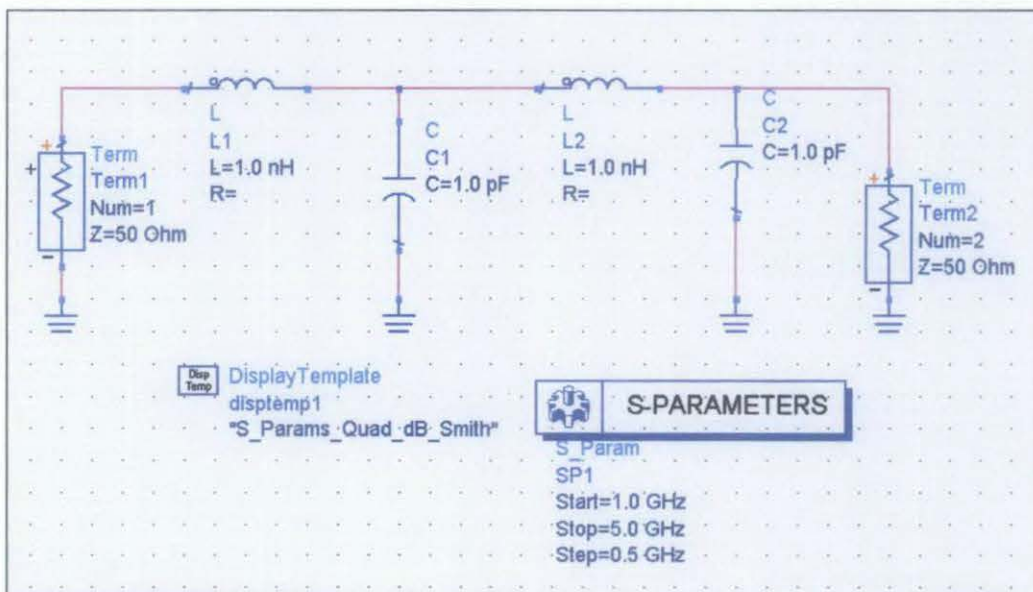


Figure 4.21: Microwave L-C Low Pass Filter Network

freq	S(1,1)	S(1,2)	S(2,1)	S(2,2)
1.000 GHz	-0.111 - j0.148	0.890 - j0.416	0.890 - j0.416	-0.042 - j0.181
1.500 GHz	-0.223 - j0.157	0.765 - j0.584	0.765 - j0.584	-0.093 - j0.256
2.000 GHz	-0.335 - j0.109	0.610 - j0.710	0.610 - j0.710	-0.158 - j0.315
2.500 GHz	-0.422 - j0.011	0.441 - j0.792	0.441 - j0.792	-0.232 - j0.353
3.000 GHz	-0.467 + j0.118	0.271 - j0.834	0.271 - j0.834	-0.309 - j0.370
3.500 GHz	-0.461 + j0.255	0.108 - j0.843	0.108 - j0.843	-0.382 - j0.363
4.000 GHz	-0.408 + j0.381	-0.044 - j0.828	-0.044 - j0.828	-0.446 - j0.335
4.500 GHz	-0.316 + j0.478	-0.188 - j0.798	-0.188 - j0.798	-0.496 - j0.287
5.000 GHz	-0.195 + j0.534	-0.329 - j0.754	-0.329 - j0.754	-0.525 - j0.220

Figure 4.22: ADS Simulation Scattering Parameters for Microwave L-C Low Pass Filter Network

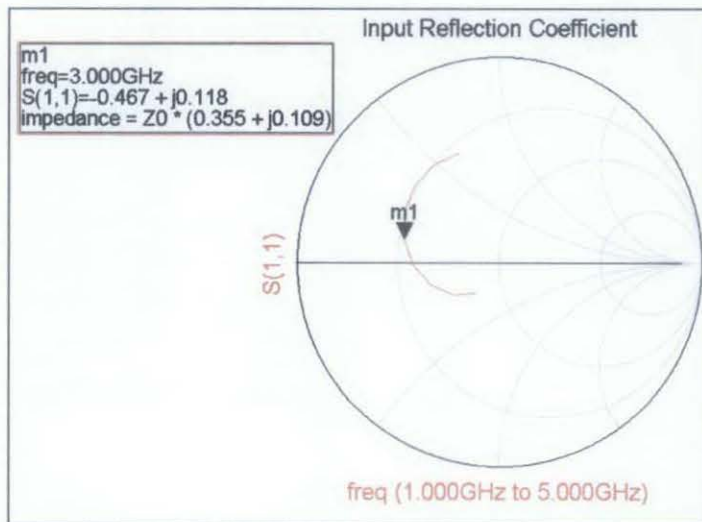


Figure 4.23: ADS Simulation Smith Chart for Microwave L-C Low Pass Filter Network

Frequency	S11	S12	S21	S22
1.0000	-0.1114 - 0.1484i	0.8901 - 0.4163i	0.8901 - 0.4163i	-0.0425 - 0.1806i
1.5000	-0.2226 - 0.1571i	0.7649 - 0.5836i	0.7649 - 0.5836i	-0.0928 - 0.2562i
2.0000	-0.3351 - 0.1087i	0.6102 - 0.7096i	0.6102 - 0.7096i	-0.1577 - 0.3151i
2.5000	-0.4225 - 0.0109i	0.4411 - 0.7917i	0.4411 - 0.7917i	-0.2317 - 0.3535i
3.0000	-0.4668 + 0.1176i	0.2708 - 0.8336i	0.2708 - 0.8336i	-0.3085 - 0.3695i
3.5000	-0.4613 + 0.2550i	0.1081 - 0.8429i	0.1081 - 0.8429i	-0.3820 - 0.3632i
4.0000	-0.4084 + 0.3807i	-0.0439 - 0.8285i	-0.0439 - 0.8285i	-0.4464 - 0.3354i
4.5000	-0.3161 + 0.4782i	-0.1875 - 0.7976i	-0.1875 - 0.7976i	-0.4961 - 0.2873i
5.0000	-0.1951 + 0.5341i	-0.3295 - 0.7537i	-0.3295 - 0.7537i	-0.5245 - 0.2195i

Figure 4.24: Matlab Scattering Parameters for Microwave L-C Low Pass Filter Network

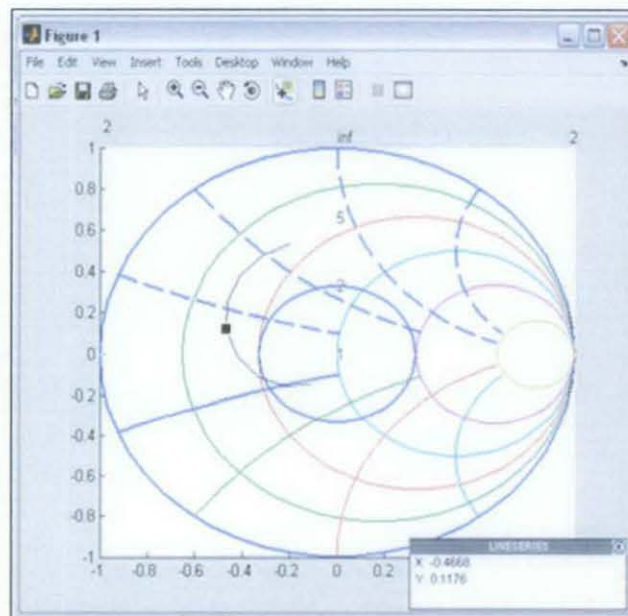


Figure 4.25: Matlab Smith Chart for Microwave L-C Low Pass Filter Network

4.1.7 Microwave L-C High Pass Filter Network

Figure 4.26 represents a microwave L-C High Pass Filter network. This filter network is considered to be a four cascaded two-port systems with the inductors and capacitors being the four cascaded systems respectively. The characteristic impedance of the network is set to 50 Ohms and the frequency of operation is within the range of 1.0 – 5.0 GHz with an increment of 0.5 GHz.

The developed Matlab program was used to enter all input data of the network's elements parameters values and ports numbers and analyze the network at the specified range of frequencies. The same network was then analyzed using ADS software package. Results of both ADS simulation and Matlab code implementation are shown clearly. Smith Chart simulation results are also clearly indicated. A value pointer is placed at 3GHz frequency point for each plot to verify the results obtained.

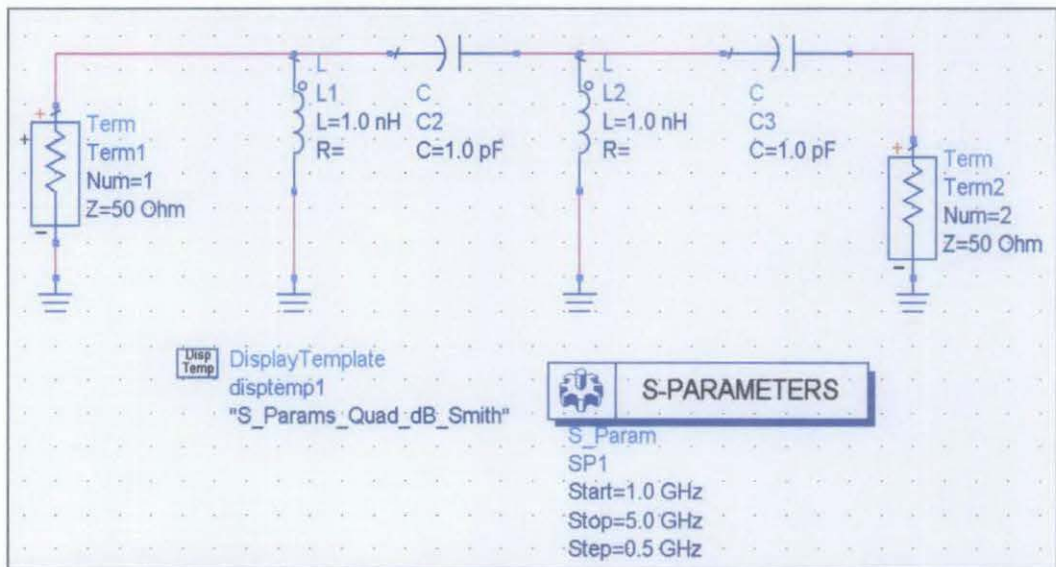


Figure 4.26: Microwave L-C High Pass Filter Network

freq	S(1,1)	S(1,2)	S(2,1)	S(2,2)
1.000 GHz	-0.966 + j0.258	0.003 - j0.001	0.003 - j0.001	0.806 - j0.592
1.500 GHz	-0.916 + j0.400	0.014 - j0.012	0.014 - j0.012	0.571 - j0.821
2.000 GHz	-0.822 + j0.566	0.039 - j0.056	0.039 - j0.056	0.244 - j0.967
2.500 GHz	-0.613 + j0.761	0.050 - j0.207	0.050 - j0.207	-0.197 - j0.957
3.000 GHz	-0.078 + j0.793	-0.244 - j0.553	-0.244 - j0.553	-0.639 - j0.476
3.500 GHz	0.064 + j0.053	-0.991 - j0.107	-0.991 - j0.107	-0.074 + j0.038
4.000 GHz	-0.381 - j0.030	-0.780 + j0.496	-0.780 + j0.496	0.134 - j0.358
4.500 GHz	-0.512 + j0.108	-0.522 + j0.673	-0.522 + j0.673	-0.023 - j0.523
5.000 GHz	-0.526 + j0.209	-0.349 + j0.747	-0.349 + j0.747	-0.178 - j0.538

Figure 4.27: ADS Simulation Scattering Parameters for Microwave L-C High Pass Filter Network

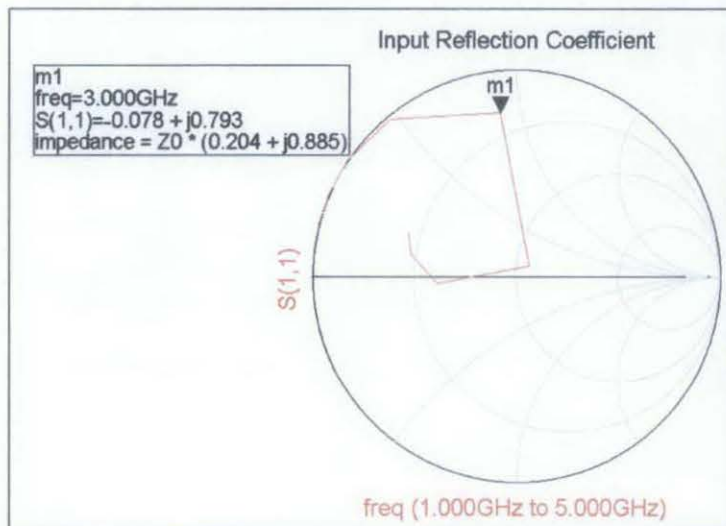


Figure 4.28: ADS Simulation Smith Chart for Microwave L-C High Pass Filter Network

Frequency	S11	S12	S21	S22
1.0000	-0.9662 + 0.2577i	0.0030 - 0.0014i	0.0030 - 0.0014i	0.8061 - 0.5918i
1.5000	-0.9161 + 0.4005i	0.0143 - 0.0117i	0.0143 - 0.0117i	0.5708 - 0.8209i
2.0000	-0.8217 + 0.5658i	0.0388 - 0.0559i	0.0388 - 0.0559i	0.2437 - 0.9675i
2.5000	-0.6128 + 0.7611i	0.0500 - 0.2065i	0.0500 - 0.2065i	-0.1969 - 0.9571i
3.0000	-0.0784 + 0.7925i	-0.2445 - 0.5532i	-0.2445 - 0.5532i	-0.6387 - 0.4756i
3.5000	0.0645 + 0.0533i	-0.9907 - 0.1071i	-0.9907 - 0.1071i	-0.0744 + 0.0383i
4.0000	-0.3807 - 0.0304i	-0.7799 + 0.4958i	-0.7799 + 0.4958i	0.1340 - 0.3577i
4.5000	-0.5122 + 0.1079i	-0.5223 + 0.6732i	-0.5223 + 0.6732i	-0.0228 - 0.5230i
5.0000	-0.5262 + 0.2091i	-0.3486 + 0.7469i	-0.3486 + 0.7469i	-0.1776 - 0.5376i

Figure 4.29: Matlab Scattering Parameters for Microwave L-C High Pass Filter Network

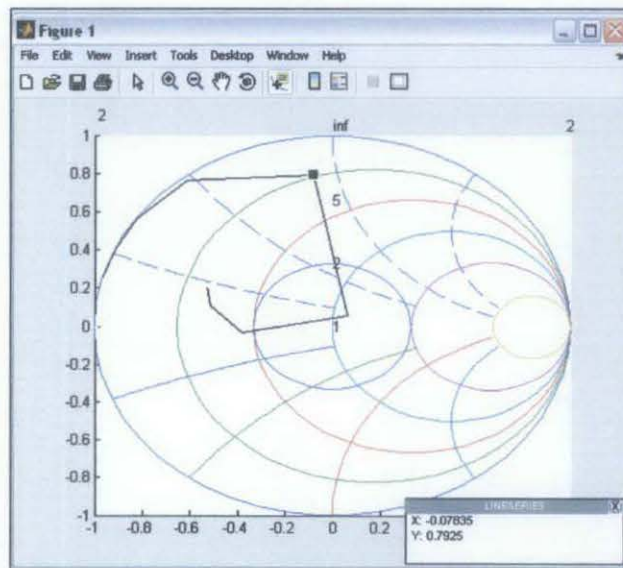


Figure 4.30: Matlab Smith Chart for Microwave L-C High Pass Filter Network

4.1.8 Microwave Microstrip Coupled-Line Band-Pass Filter

Figure 4.31 represents a microwave Coupled-Line Band Pass Filter network. This filter network is considered to be a four cascaded two-port systems with the coupled-line filter sections being the four cascaded systems respectively. The characteristic impedance of the network is set to 50 Ohms and the frequency of operation is within the range of 0.7– 1.0 GHz with an increment of 0.02 GHz.

The developed Matlab program was used to enter all input data of the network's elements parameters values and ports numbers and analyze the network at the specified range of frequencies. The same network was then analyzed using ADS software package. Results of both ADS simulation and Matlab code implementation are shown clearly. Smith Chart simulation results are also clearly indicated. A value pointer is placed at 0.74 GHz frequency point for each plot to verify the results obtained.

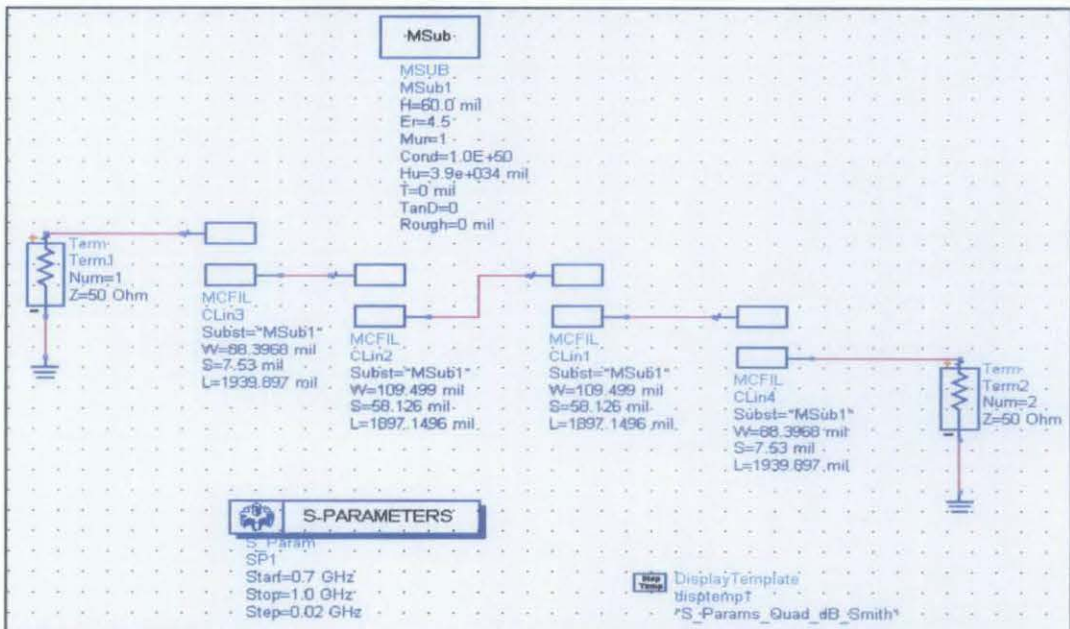


Figure 4.31: Microstrip Coupled-Line Band-Pass Filter Network

freq	S(1,1)	S(1,2)	S(2,1)	S(2,2)
700.0 MHz	-0.997 + j0.068	-0.002 - j0.032	-0.002 - j0.032	-0.997 + j0.068
720.0 MHz	-0.959 + j0.277	-0.015 - j0.054	-0.015 - j0.054	-0.959 + j0.277
740.0 MHz	-0.840 + j0.531	-0.058 - j0.091	-0.058 - j0.091	-0.840 + j0.531
760.0 MHz	-0.526 + j0.815	-0.204 - j0.132	-0.204 - j0.132	-0.526 + j0.815
780.0 MHz	0.199 + j0.764	-0.593 + j0.155	-0.593 + j0.155	0.199 + j0.764
800.0 MHz	0.210 + j5.782E-4	-0.003 + j0.978	-0.003 + j0.978	0.210 + j5.782E-4
820.0 MHz	0.019 - j0.035	0.873 + j0.485	0.873 + j0.485	0.019 - j0.035
840.0 MHz	-0.062 - j0.144	0.906 - j0.392	0.906 - j0.392	-0.062 - j0.144
860.0 MHz	-0.081 - j0.021	0.249 - j0.965	0.249 - j0.965	-0.081 - j0.021
880.0 MHz	0.302 - j0.387	-0.687 - j0.536	-0.687 - j0.536	0.302 - j0.387
900.0 MHz	-0.291 - j0.852	-0.412 + j0.141	-0.412 + j0.141	-0.291 - j0.852
920.0 MHz	-0.733 - j0.647	-0.140 + j0.158	-0.140 + j0.158	-0.733 - j0.647
940.0 MHz	-0.911 - j0.395	-0.047 + j0.109	-0.047 + j0.109	-0.911 - j0.395
960.0 MHz	-0.981 - j0.180	-0.014 + j0.074	-0.014 + j0.074	-0.981 - j0.180
980.0 MHz	-0.999 + j0.002	8.246E-5 + j0.051	8.246E-5 + j0.051	-0.999 + j0.002
1.000 GHz	-0.987 + j0.157	0.006 + j0.037	0.006 + j0.037	-0.987 + j0.157

Figure 4.32: ADS Simulation Scattering Parameters for Microwave Coupled-Line Band Pass Filter Network

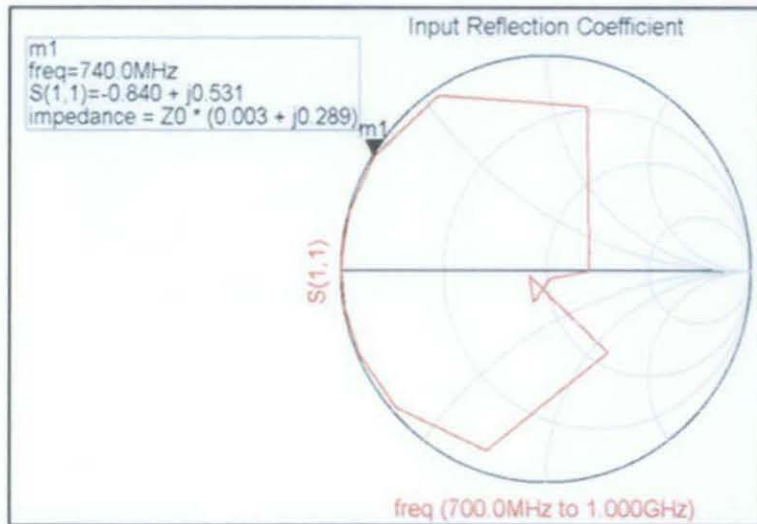


Figure 4.33: ADS Simulation Smith Chart for Microwave Coupled-Line Band Pass Filter Network

Frequency	S11	S12	S21	S22
0.7000	-0.9980 + 0.0523i	-0.0019 - 0.0364i	-0.0019 - 0.0364i	-0.9980 + 0.0524i
0.7200	-0.9680 + 0.2443i	-0.0142 - 0.0564i	-0.0142 - 0.0564i	-0.9679 + 0.2443i
0.7400	-0.8757 + 0.4723i	-0.0479 - 0.0888i	-0.0479 - 0.0888i	-0.8756 + 0.4723i
0.7600	-0.6503 + 0.7337i	-0.1475 - 0.1307i	-0.1475 - 0.1307i	-0.6502 + 0.7338i
0.7800	-0.1171 + 0.8897i	-0.4376 - 0.0575i	-0.4376 - 0.0575i	-0.1169 + 0.8897i
0.8000	0.3950 + 0.2794i	-0.5053 + 0.7146i	-0.5053 + 0.7146i	0.3951 + 0.2793i
0.8200	0.0521 - 0.0311i	0.5123 + 0.8567i	0.5123 + 0.8567i	0.0521 - 0.0312i
0.8400	0.0081 - 0.0883i	0.9919 + 0.0909i	0.9919 + 0.0909i	0.0081 - 0.0883i
0.8600	-0.1216 - 0.1167i	0.6826 - 0.7111i	0.6826 - 0.7111i	-0.1216 - 0.1168i
0.8800	0.1523 - 0.0452i	-0.2810 - 0.9465i	-0.2810 - 0.9465i	0.1523 - 0.0453i
0.9000	0.0095 - 0.8093i	-0.5872 - 0.0068i	-0.5872 - 0.0068i	0.0094 - 0.8093i
0.9200	-0.6318 - 0.7354i	-0.1858 + 0.1596i	-0.1858 + 0.1596i	-0.6319 - 0.7353i
0.9400	-0.8829 - 0.4544i	-0.0542 + 0.1053i	-0.0542 + 0.1053i	-0.8830 - 0.4543i
0.9600	-0.9746 - 0.2139i	-0.0141 + 0.0643i	-0.0141 + 0.0643i	-0.9746 - 0.2139i
0.9800	-0.9991 - 0.0162i	-0.0007 + 0.0403i	-0.0007 + 0.0403i	-0.9991 - 0.0162i
1.0000	-0.9884 + 0.1499i	0.0040 + 0.0261i	0.0040 + 0.0261i	-0.9883 + 0.1499i

Figure 4.34: Matlab Scattering Parameters for Microwave Coupled-Line Band Pass Filter Network

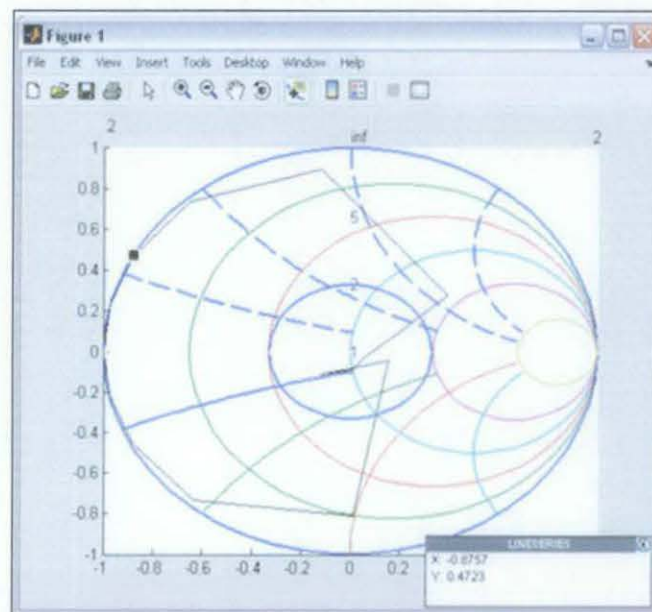


Figure 4.35: Matlab Smith Chart for Microwave Coupled-Line Band Pass Filter Network

Vector S21 obtained from the results shown in Figure 4.34 was extracted from the results and was used to plot a linear graph of the forward transmission response of the network shown in Figure 4.31 by plotting the magnitude of S21 over the frequency range given. The graph plotted was compared to the graph obtained from ADS simulation of the network and the results are shown below:

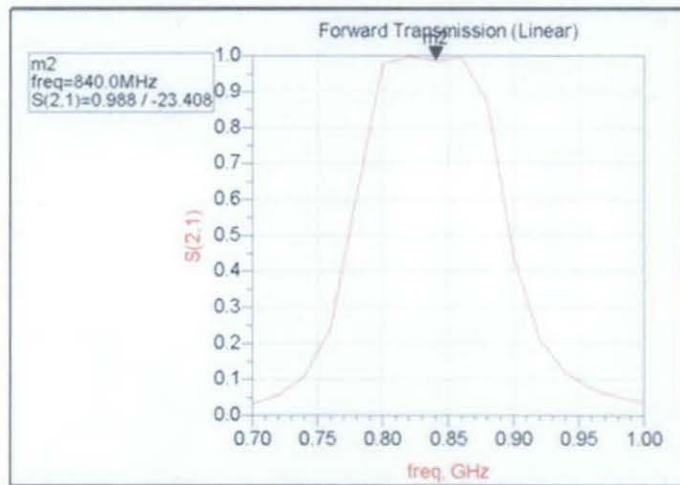


Figure 4.36: ADS Forward Transmission Simulation for Microwave Coupled-Line Band Pass Filter Network

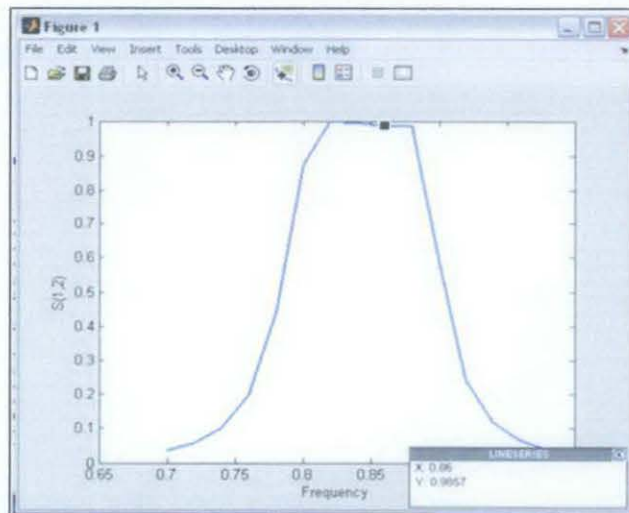


Figure 4.37: Matlab Forward Transmission Simulation for Microwave Coupled-Line Band Pass Filter Network

4.2.2 Analysis of Microwave Filter Networks

The developed Matlab program was used to analyze and simulate some microwave networks. For the purpose of this project, three microwave networks were analyzed; Microwave L-C Low Pass Filter, Microwave L-C High Pass Filter, and Microwave Microstrip Coupled-Line Band Pass Filter. Comparison of results obtained from ADS simulation and Matlab code implementation of these networks indicates that the results are very similar to each other with an acceptable range of accuracy.

The results obtained from Matlab program analysis of Microwave L-C Low Pass and L-C High Pass Filter Networks are almost the same as the results obtained from ADS analysis of the same networks. On the other hand, the results obtained from Matlab program analysis of Microwave Microstrip Coupled-Line Band Pass Filter Network differ a little from the results obtained from ADS analysis. This is because the mathematical model used to implement the Matlab code in this project may differ a bit from the ideal model ADS software uses to analyze Coupled-Line Filter Sections. This is due to different mathematical models available to analyze microstrip elements as explained earlier. However, careful comparison of both results shows an acceptable range of similarity and accuracy between them.

From all the results obtained and shown clearly in the previous section, it can be concluded that the results obtained from implementation of the Matlab code developed in this project are very similar to the results obtained from ADS simulation. This proves that the method chosen to analyze microwave networks; the Connection-Scattering Matrix method; is effective and reliable.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

A detailed and practical computer code that acts as basis for an alternative Computer Aided Design (CAD) program for microwave network analysis and design is successfully developed in this final year project. This alternative CAD program is to be specially developed to fulfill the requirements and learning outcomes of Microwave Engineering courses at universities and engineering institutions. The computer code required in this final year project is developed based on the Connection-Scattering Matrix method and implemented via Matlab 2007 software package. A detailed verification of the results obtained is successfully achieved through comparison with the results obtained from Advanced Design System (ADS) software packages used for analysis and simulation of microwave networks.

The developed Matlab code successfully achieved the main objectives of this final year project as follows:

- i. Successfully performs computerized analysis and simulation of microwave networks with accurate and reliable results.
- ii. Incorporates a suitable Graphical User Interface (GUI) for an easy and convenient usage and implementation of the developed code.
- iii. It is an open-source code that students can easily access to modify for further development.

5.2 Recommendations

This final year project successfully developed an open source Matlab code that can be used as a basis for an alternative CAD program for microwave network analysis and simulation. This Program can be implemented to fulfill the requirements and learning outcomes of Microwave Engineering courses at universities and engineering institutions as an alternative to the expensive available CAD programs currently being used. It is recommended that students access and modify the developed code for further enhancement of the program. There are several areas of improvements can be implemented on the developed code such as:

5.2.1 Number of Network Components

It is recommended that the developed code program be further improved by including the option of using more than six components to analyze any given microwave network. The program can include the option of 'Add Component' whenever the user needs to analyze microwave networks that contain more than six network components.

5.2.2 Analysis of Other Microwave Elements

The developed program allows the user to select a network component from a menu that contains five different microwave elements. It is recommended that the program be further improved by including the analysis of microwave elements such as stripline elements and other lumped and microstrip elements.

5.2.3 Simulation of Microwave Networks

The developed program allows the user to perform network simulation by plotting the Smith Chart of the input reflection coefficients; S_{11} vector. The program can be further developed by developing Matlab codes to plot the network response in terms of S_{21} and S_{12} vectors and include these plots in the GUI developed in this project. These plots can be very useful in design and analysis of microwave networks.

REFERENCES

- [1] Janusz A. Dobrowolski & Wojciech Ostrowski, *Computer-Aided Analysis, Modeling and Design of Microwave Networks, The Wave Approach*. Artech House, Boston, London, 1996.
- [2] Vito A. Monaco, *Computer-Aided Analysis of Microwave Circuits*, IEEE Transactions on Microwave theory and Techniques, Vol. MTT-22, No. 3, March 1974.
- [3] Less Besser, Rowan Gilmore, *Practical RF Circuit Design for Modern Wireless Systems*, volume I, Passive Circuits and Systems, Artech House, Boston, London, 2003.
- [4] <http://www.microwaves101.com/encyclopedia/historyCAD.cfm>
- [5] Guillermo Gonzalez, *Microwave Transistor Amplifiers, Analysis and Design*, 2nd Edition, Prentice Hall, Inc. © 1997.
- [6] Gupta, *Computer Aided Design of Microwave Circuits*, Artech House, Boston, London, 1981, chapters 11, 19 and 20.
- [7] Gunter Kompa, *Practical Microstrip Design and Applications*, Artech House, Boston, London, 2005.
- [8] Mongia, Bahl, Bhartia, Hong, *RF and Microwave Coupled-Line Circuits*, 2nd Edition, Artech House, Boston, London, 2007.

APPENDIX A

➤ Subroutines to Calculate Scattering Parameters for a Series Inductor:

```
%%% Function to calculate Scattering Parameters of a Series Inductor
%%% for a given range of frequencies

function s_indc_series=s_all_indc_series(Z0,L,f1,f2,incrt)
f1 = f1*10^9;           % Start frequency in GHz
f2 = f2*10^9;           % Stop frequency in GHz
incrt = incrt*10^9;     % frequency increment value in GHz
L = L*10^-9;           % Inductor value in nH
s_indc_series=[];
for i=f1:incrt:f2       % Loop repeats based on increment in frequency
    k=INDC_SERIES(Z0,i,L); % calculate Scattering Parameters for one
                          % frequency at a time
    s_indc_series=[s_indc_series;k];
end

function LM = INDC_SERIES(Z0,f,L)
w = 2 * pi * f;        % 'f' is the frequency of operation
Z = w * L * i;         % 'L' is the inductor value in nH
SL11 = Z / (Z + 2 * Z0); % 'Z0' is the characteristic impedance
SL12 = 2 * Z0 / (Z + 2 * Z0);
SL22 = SL11;
SL21 = SL12;
LM = [SL11 SL12 SL21 SL22];
```

➤ Subroutines to Calculate Scattering Parameters for a Shunt Inductor:

```
%%% Function to calculate Scattering Parameters of a Shunt Inductor
%%% for a given range of frequencies

function s_indc_shunt=s_all_indc_shunt(Z0,L,f1,f2,incrt)
f1 = f1*10^9;           % Start frequency in GHz
f2 = f2*10^9;           % Stop frequency in GHz
incrt = incrt*10^9;     % frequency increment value in GHz
L = L*10^-9;           % Inductor value in nH
s_indc_shunt=[];
for i=f1:incrt:f2       % Loop repeats based on increment in frequency
    k=INDC_SHUNT(Z0,i,L); % calculate Scattering Parameters for one
                          % frequency at a time
    s_indc_shunt=[s_indc_shunt;k];
end

function LM1 = INDC_SHUNT(Z0,f,L)
w = 2 * pi * f;        % 'f' is the frequency of operation
Z = w * L * i;         % 'L' is the inductor value in nH
Y = 1/Z;
SL11 = -Z0*Y/(2+ Z0*Y); % 'Z0' is the characteristic impedance
SL12 = 2 / (2 + Z0*Y);
SL22 = SL11;
SL21 = SL12;
LM1 = [SL11 SL12 SL21 SL22];
```


➤ **Subroutines to Calculate Scattering Parameters for a Single Microstrip Coupled-Line Filter Section:**

```
%%% Function to calculate Scattering Parameters of a single microstrip
%%% coupled line filter section for a given range of frequencies
```

```
function s_coupled=s_all_coupled(input_coupled,f1,f2,incrt)
W= input_coupled(1);           % width of the substrate
h =input_coupled(2);           % height of the substrate
L=input_coupled(3);            % Length of the inductor
s=input_coupled(4);            % separation between conductors
Er= input_coupled(5);           % dielectric constant of the substrate material
f1 = f1*10^9;                   % start frequency in GHz
f2 = f2*10^9;                   % stop frequency in GHz
incrt = incrt*10^9;             % increment in frequency in GHz
s_coupled=[];
for i=f1:incrt:f2               % loop repeats based on the frequency increment
    k=Coupled(h,W,s,L,i,Er);
    s_coupled=[s_coupled;k];
end
```

```
function [S] = Coupled(h,W,s,L,f,Er)
ERE = Dielec (W,Er,h);          % calculate the effective dielectric constant
Zu = IMPWU(ERE,W,h);            % calculate the impedance based on ERE
Eree = Dielec_even (W,Er,h,s,ERE); % calculate even-mode impedance
Ereo = Dielec_odd (W,Er,h,s,ERE); % calculate the odd-mode impedance
z =IMPEO(W,h,s,Er,ERE,Eree,Ereo,Zu); % calculate the characteristic impedance
c = 0.3*10^9;                   % speed of light
L = L*2.54*10^-5;               % length of the conductor
wp = 2.0*pi*f;
Beta = wp* ((sqrt(ERE))/c);
teta = Beta *L;                  % electrical length of the coupled line
Z = Z_param(z(1),z(2),teta);     % calculate the Z-parameters of the line
Z0 =50;
S = conZ2S(Z(1),Z(2),Z(3),Z(4),Z0); % calculate the S-parameters of the line
```

```
%%% Function to calculate the Effective Dielectric Constant based on
%%% given dielectric constant and substrate height and width
```

```
function ERE = Dielec (W,Er,h); % 'W' is the substrate width
u = W/h;                        % 'h' is the substrate height
A1 = 1 + (1/49)* log((u^4 + (u/52)^2)/ (u^4 + 0.432));
A2 = (1/18.7)*log (1+ (u/18.1)^3);
A = A1 +A2;
B = 0.564 *((Er - 0.9)/(Er + 3))^0.053; % 'Er' is the dielectric constant
FWh =(1+ (10/u))^ (-A*B);
ERE = ((Er+1)/2)+((Er - 1)/2)*FWh;
```

```
%%% Function to calculate the characteristic impedance from known ERE, width
%%% and height of the substrate
```

```
function Zu = IMPWU(ERE,W,h);
l = W/h;
fu = 6 + ((2*pi)-6)* exp (-1*(30.666/u)^0.7528);
Zu = (60/sqrt(ERE)) * (log ((fu/u) + sqrt (1 + (2/u)^2)));
```

```

%%% Function to calculate the Effective Dielectric Constant for the even mode
function Eree = Dielec_even (W,Er,h,s,ERE);
u = W/h;
g = s/h; %'s' is the separation between conductors
v = (u *(20 + g^2)/(10 + g^2))+ g*exp(-g);
aev = 1 +(log(((v^4)+(v/(52)^2))/((v^4)+0.432))/49)+(log(1+(v/18.1)^3)/18.7);
bev = 0.564 *((Er-0.9)/(Er+3))^0.053;
Eree = 0.5*(Er+1) +0.5*(Er-1)*(1+ (10/v))^ (-1*aev*bev);

%%% Function to calculate the Effective Dielectric Constant for the odd mode
function Ereo = Dielec_odd (W,Er,h,s,ERE);
u = W/h;
g = s/h;
aod = 0.7287 * (ERE-0.5*(Er+1))*(1-exp(-0.179*u));
bod = (0.747 *Er)/(0.15+Er);
cod = bod-(bod-0.207)*exp(-0.414*u);
dod = 0.593+0.694*exp(-0.562*u);
Ereo = (0.5*(Er+1)+aod-ERE)*exp(-cod*g^dod)+ ERE;

%%% Function to calculate the even and odd modes impedances
function [Zm] = IMPEO(W,h,s,Er,ERE,Eree,Ereo,Zu);
u = W/h;
g = s/h;
Q1 = 0.8695*(u)^0.194;
Q2 = 1+0.7519*g+0.189*(g)^2.31;
Q3 = 0.1975+(16.6+(8.4/g)^6)^-0.387 +log((g^10)/(1+(g/3.4)^10))/241;
Q4 = (2*Q1/Q2)* (exp(-g)*(u^Q3)+(2-exp(-g))*(u^-Q3))^-1;
Ze = Zu* (sqrt(ERE/Eree))* (1/(1-(Zu/377)* ((ERE)^0.5)*Q4));

Q5 = 1.794+1.14*log(1+(0.638/(g+0.517*(g)^2.43)));
Q6 = 0.2305+log((g)^10/(1+(g/5.8)^10))/281.3+log(1+0.598*g^1.154)/5.1;
Q7 = (10+190*(g)^2)/(1+82.3*(g)^3);
Q8 = exp(-6.5 -0.95*log(g)-(g/0.15)^5);
Q9 = log(Q7)*(Q8+ (1/16.5));
Q10 = ((Q2)^-1)*(Q2*Q4-Q5*exp(log(u)*Q6*(u)^-Q9));
Zo = Zu* (sqrt(ERE/Ereo))* (1/(1-(Zu/377)* ((ERE)^0.5)*Q10));
Zm = [Ze Zo];

%%% Function to calculate the Z-parameters of single coupled line filter
%%% section from the known Ze and Zo
function [Z] = Z_param(Ze,Zo,teta);
Z0 = sqrt (Ze*Zo);
Z11 = (-i/2)*(Ze +Zo)*cot (teta);
Z22 = Z11;
Z12 = (-i/2)*(Ze -Zo)*csc (teta);
Z21 = Z12;
Z = [Z11 Z12 Z21 Z22 Z0];

%%% Function to convert the Z-parameters to Scattering Parameters
function [Sz] = conZ2S(Z11,Z12,Z21,Z22,Z0);
Zdelta = (Z11 + Z0)*(Z22 +Z0)- Z12*Z21;
S11 = ((Z11 - Z0)*(Z22 + Z0) - Z12*Z21)/Zdelta;
S12 = (2.0*Z12*Z0)/Zdelta;
S21 = (2.0*Z21*Z0)/Zdelta;

```

```
S22 = ((Z11 + Z0)*(Z22 - Z0) - Z12*Z21)/Zdelta;
Sz = [S11 S12 S21 S22];
```

► Subroutines to Calculate Scattering Parameters for a Single Microstrip Tee Junction:

```
%%% Function to calculate Scattering Parameters of a single microstrip tee
%%% junction for a given range of frequencies
```

```
function s_tee=s_all_tee(input_tee,f1,f2,incrt)
W1= input_tee(1);           % width of the main substrate
W2= input_tee(2);           % width of the second substrate
h= input_tee(3);           % height of the substrate
Er= input_tee(4);           % dielectric constant of the substrate material
f1 = f1*10^9;               % start frequency in GHz
f2 = f2*10^9;               % stop frequency in GHz
incrt = incrt*10^9;         % increment frequency in GHz
s_tee=[];
for i=f1:incrt:f2           % loops repeats based on frequency increment
    k=tee_junc(W1,W2,Er,h,i);
    s_tee=[s_tee;k];
end
```

```
%%% Function to calculate Scattering Parameters for microstrip Tee Junction
```

```
function [S] = tee_junc (W1,W2,Er,h,f);
W1 = W1*(2.54*10^-5);       %%% put it in meters
W2 = W2*(2.54*10^-5);
h = h*(2.54*10^-5);
u1 = W1/h;
A11 = 1 + (1/49)* log((u1^4 + (u1/52)^2)/ (u1^4 + 0.432));
A12 = (1/18.7)*log (1+ (u1/18.1)^3);
A1 = A11 +A12;
B1 = 0.564 *((Er - 0.9)/(Er + 3))^0.053;
FWH1 =(1+ (10/u1))^ (-A1*B1);
ERE1 = ((Er+1)/2)+((Er - 1)/2)*FWH1;
fu1 = 6 + ((2*pi)-6)* exp (-1*(30.666/u1)^0.7528);
Zu1 = (60/sqrt(ERE1)) * (log ((fu1/u1) + sqrt (1 + (2/u1)^2)));

u2 = W2/h;
A21 = 1 + (1/49)* log((u2^4 + (u2/52)^2)/ (u2^4 + 0.432));
A22 = (1/18.7)*log (1+ (u2/18.1)^3);
A2 = A21 +A22;
B2 = 0.564 *((Er - 0.9)/(Er + 3))^0.053;
FWH2 =(1+ (10/u2))^ (-A2*B2);
ERE2 = ((Er+1)/2)+((Er - 1)/2)*FWH2;
fu2 = 6 + ((2*pi)-6)* exp (-1*(30.666/u2)^0.7528);
Zu2 = (60/sqrt(ERE2)) * (log ((fu2/u2) + sqrt (1 + (2/u2)^2)));

Z0 = 50;
c = 0.3*10^9;               %%% speed of light
Lw1 = (Zu1*sqrt (ERE1))/c;
Lw2 = (Zu2*sqrt (ERE2))/c;

Ct = (((100/tanh(0.0072*Zu2))+0.64*Zu2-261)*W1);           %%% in pF
Ll = (-W2/h*(W2/h)*(-0.016*(W1/h)+0.064)+0.016/(W1/h))*Lw1)*h; %%% in nH
```

```
L2 = (((0.12*(W1/h)-0.47)*(W2/h)+0.195*(W1/h)-0.357+0.0283*sin(pi*(W1/h)-
0.75*pi))*Lw2)*h;
```

```
X1 = i*2*pi*f*L1/Z0;
```

```
X2 = i*2*pi*f*L2/Z0;
```

```
X3 = 1/i*2*pi*f*Z0*Ct;
```

```
Zin1 = ((1+X1)*(1+X2)*(1+X3))/((1+X1)*(1+X1+X2+X3)+((1+X2)*(X1+X3)));
```

```
Zin2 = ((1+X1)*(X2+X3))/(1+X1+(2*(X2+X3)));
```

```
S11 = (Zin1 - 1)/(Zin1+1);
```

```
S22 = S11;
```

```
S12 = ((1-S11)*X3*(1+X2))/((1+X2)*(1+X1+X3)+X3*(1+X1));
```

```
S21 = S12;
```

```
S13 = ((1-S11)*X3*(1+X1))/((1+X2)*(1+X1+X3)+X3*(1+X1));
```

```
S31 = S13;
```

```
S23 = S13;
```

```
S32 = S23;
```

```
S33 = (Zin2 - 1)/(Zin2+1);
```

```
[S] = [S11 S12 S13 S21 S22 S23 S31 S32 S33];
```

➤ Subroutines to Calculate Scattering Parameters for a Single Open-End Microstrip Transmission Line:

```
%%% Function to calculate Scattering Parameters of a single microstrip
%%% open-end transmission line for a given range of frequencies
```

```
function s_open=s_all_open(input_line,f1,f2,incrt)
```

```
W= input_line(1); % Width of the substrate
```

```
h= input_line(2); % height of the substrate
```

```
L= input_line(3); % length of the conductor
```

```
Er= input_line(4); % dielectric constant of the substrate material
```

```
f1 = f1*10^9; % start frequency in GHz
```

```
f2 = f2*10^9; % stop frequency in GHz
```

```
incrt = incrt*10^9; % frequency increment in GHz
```

```
s_open=[];
```

```
for i=f1:incrt:f2 % loop repeats based on the frequency increment
```

```
    k=open_end1(W,h,L,Er,i);
```

```
    s_open=[s_open;k];
```

```
end
```

```
%%% Function to calculate Scattering Parameters for an Open-End Microstrip
%%% Transmission Line
```

```
function S11 = open_end1 (W,h,L,Er,f);
```

```
u = W/h;
```

```
A1 = 1 + (1/49)* log((u^4 + (u/52)^2)/ (u^4 + 0.432));
```

```
A2 = (1/18.7)*log (1+ (u/18.1)^3);
```

```
A = A1 +A2;
```

```
B = 0.564 *((Er - 0.9)/(Er + 3))^0.053;
```

```
FWh =(1+ (10/u))^ (-A*B);
```

```
ERE = ((Er+1)/2)+((Er - 1)/2)*FWh;
```

```
fu = 6 + ((2*pi)-6)* exp (-1*(30.666/u)^0.7528);
```

```
Zu = (60/sqrt(ERE)) * (log ((fu/u) + sqrt (1 + (2/u)^2)));
```

```
c = 0.3*10^9; %%% speed of light
```

```
Z0 =50;
```

```
h = h*(2.54 *10^-5);
```

```

L = L*(2.54 *10^-5);
DL = (0.412*h*(ERE+0.3)*(u+0.264))/((ERE-0.258)*(u+0.813))+L;
BL = DL*2*pi*f*sqrt(ERE)/c;
Zin = - i* Zu*cot(BL);
S11 = (Zin-Z0)/(Zin+Z0);

```

➤ Subroutines to Set Input Data Files for all Network Elements:

```

%%% Function to initialize all input data files for all elements

function initial_all
findc = fopen('INDC_SERIES_data','w'); % series inductor input data file
fclose(findc);
findcl = fopen('INDC_SHUNT_data','w'); % shunt inductor input data file
fclose(findcl);
fcap = fopen('CAP_SERIES_data','w'); % series capacitor input data file
fclose(fcap);
fcapl = fopen('CAP_SHUNT_data','w'); % shunt capacitor input data file
fclose(fcapl);
fcoupled=fopen('Coupled_data','w'); % coupled line input data file
fclose(fcoupled);
fjunc = fopen('T_junction_data','w'); % tee junction input data file
fclose(fjunc);
fline = fopen('Open_End_data','w'); % open-end line input data file
fclose(fline);

```

1. Series Inductor

```

%%% Function to create input data set for all series inductor elements
%%% selected by the user for n number of times

function inductor_series_data (indcl)
findc_series = fopen('INDC_SERIES_data','a'); % open file and append data
                                                    % from the end without overwriting
fprintf(findc_series,' %g',indcl);
fclose(findc_series);

%%% Function to read input data for all series inductor elements
function inductor_series =inductor_series_read ()
findcl = fopen('INDC_SERIES_data');
inductor_series =fscanf(findcl,'%f');
fclose(findcl);

```

2. Shunt Inductor

```

%%% Function to create input data set for all shunt inductor elements
%%% selected by the user for n number of times

function inductor_shunt_data (indc2)
findc_shunt = fopen('INDC_SHUNT_data','a'); % open file and append data
                                                    % from the end without overwriting

fprintf(findc_shunt,' %g',indc2);
fclose(findc_shunt);

%%% Function to read input data for all shunt inductor elements

```

```

function inductor_shunt =inductor_shunt_read ()
findc2 = fopen('INDC_SHUNT_data');
inductor_shunt =fscanf(findc2,'%f');
fclose(findc2);

```

3. Series Capacitor

```

%%% Function to create input data set for all series capacitor elements
%%% selected by the user for n number of times

```

```

function capacitor_series_data (cap2)
fcap_series = fopen('CAP_SERIES_data','a'); % open file and append data
                                                    % from the end without overwriting

```

```

fprintf(fcap_series, ' %g', cap2);
fclose(fcap_series);

```

```

%%% Function to read input data for all series capacitor elements
function capacitor_series = capacitor_series_read()
fcap2 = fopen('CAP_SERIES_data');
capacitor_series=fscanf(fcap2,'%f');
fclose(fcap2);

```

4. Shunt Capacitor

```

%%% Function to create input data set for all shunt capacitor elements
%%% selected by the user for n number of times

```

```

function capacitor_shunt_data (cap1)
fcap_shunt = fopen('CAP_SHUNT_data','a'); % open file and append data
                                                    % from the end without overwriting

```

```

fprintf(fcap_shunt, ' %g', cap1);
fclose(fcap_shunt);

```

```

%%% Function to read input data for all shunt capacitor elements
function capacitor_shunt = capacitor_shunt_read()
fcap1 = fopen('CAP_SHUNT_data');
capacitor_shunt=fscanf(fcap1,'%f');
fclose(fcap1);

```

5. Microstrip Coupled Line Filter Section

```

%%% Function to create input data set for all coupled line filter sections
%%% selected by the user for n number of times

```

```

function coupled_data(Width,Height,Length,Seperation,D_constant)
fcoupled = fopen('Coupled_data','a');
fprintf(fcoupled, '\n %g %g %g %g
%g',Width,Height,Length,Seperation,D_constant);
fclose(fcoupled);

```

```

%%% Function to read input data from user for each coupled line sections
function coupled = coupled_read()
fcoupled1 = fopen('Coupled_data');
coupled=[];
for i=1:10
    if( feof(fcoupled1)) % if no data stored in the data file

```

```

        break
    end
)coupled1=fscanf(fcoupled1,'%g %g %g %g %g',[1 5]);
Coupled=[Coupled;Dcoupled1];
end
fclose(fcoupled1);

```

6. Microstrip Tee Junction

```

%%% Function to create input data set for all tee junction elements selected
%%% by the user for n number of times

```

```

function tee_junc_data(Width1,Width2,Height,D_constant)
fjunc = fopen('T_junction_data','a');
fprintf(fjunc,'\n %g %g %g %g',Width1,Width2,Height,D_constant);
fclose(fjunc);

```

```

%%% Function to read input data from user for each tee_junction element

```

```

function tee_junc= tee_junc_read ()
fjunc1 = fopen('T_junction_data')
T_junction=[];
for i=1:10
    if( feof(fjunc1))          % if no data stored in the data file
        break
    end
Djunc1=fscanf(fjunc1,'%g %g %g %g',[1 4]);
T_junction=[T_junction;Djunc1];
end
fclose(fjunc1);

```

7. Microstrip Open-End Transmission Line

```

%%% Function to create input data set for all open_end transmission line
%%% sections selected by the user for n number of times

```

```

function open_end_data(Width,Height,Length,D_constant)
fline = fopen('Open_End_data','a');
fprintf(fline,'\n %g %g %g %g',Width,Height,Length,D_constant);
fclose(fline);

```

```

%%% Function to read input data from user for each open_end line elements

```

```

function line = open_end_read()
fline1 = fopen('Open_End_data');
Open_End=[];
for i=1:10
    if( feof(fline1))          % if no data stored in the data file
        break
    end
Dline1=fscanf(fline1,'\ %g %g %g %g',[1 4]);
Open_End=[Open_End;Dline1];
end
fclose(fline1);

```

➤ Subroutine to Set Number of Ports for all Network Elements:

```

%%% Function to determine the element's number of ports when selected from
%%% the menu by the user

```



```

    element_times(1)=element_times(1)+1;
%indicates that series inductor has been chosen more than one time
    Ep=[Ep 2];    % element has 2 ports

    case 3
        inductor_shunt=inductor_shunt_read ();
        temp =
s_all_indc_shunt(Z0,inductor_shunt(element_times(2)),f1,f2,incrt);
        S_indc_shunt=[S_indc_shunt; temp];
        element_times(2)=element_times(2)+1;
%indicates that shunt inductor has been chosen more than one time
        Ep=[Ep 2];    % element has 2 ports

    case 4
        capacitor_series = capacitor_series_read();
        temp =
s_all_cap_series(Z0,capacitor_series(element_times(3)),f1,f2,incrt);
        S_cap_series=[S_cap_series; temp];
        element_times(3)=element_times(3)+1;
%indicates that series capacitor has been chosen more than one time
        Ep=[Ep 2];    % element has 2 ports

    case 5
        capacitor_shunt = capacitor_shunt_read();
        temp =
s_all_cap_shunt(Z0,capacitor_shunt(element_times(4)),f1,f2,incrt);
        S_cap_shunt=[S_cap_shunt; temp];
        element_times(4)=element_times(4)+1;
%indicates that shunt capacitor has been chosen more than one time
        Ep=[Ep 2];    % element has 2 ports

    case 6
        coupled = coupled_read ();
        temp =s_all_coupled(coupled(element_times(5),:),f1,f2,incrt);
        S_coupled=[S_coupled; temp];
        element_times(5)=element_times(5)+1;
%indicates that coupled line has been chosen more than one time
        Ep=[Ep 2];    % element has 2 ports

    case 7
        tee_junc= tee_junc_read ();
        temp = s_all_tee(tee_junc(element_times(6),:),f1,f2,incrt);
        S_junc=[S_junc; temp];
        element_times(6)=element_times(6)+1;
%indicates that tee junction has been chosen more than one time
        Ep=[Ep 3]; % element has 3 ports

    case 8
        line =open_end_read ();
        temp =s_all_open(line(element_times(7),:),f1,f2,incrt);
        S_line=[S_line; temp];
        element_times(7)=element_times(7)+1;
%indicates that open-end line has been chosen more than one time
        Ep=[Ep 1];    % element has 1 port

```

```

        end
end

Sp_all=[];
freq_range = size((f1:incrt:f2),2); %frequency range given

for i=1:freq_range
    temp_indc_series=0;
    temp_indc_shunt=0;
    temp_cap_series=0;
    temp_cap_shunt=0;
    temp_coupled=0;
    temp_tee=0;
    temp_line=0;
    Sp=[];
    for j =1:size(Ep,2) % the no of elements
        switch menus(j) % to follow the elements in order specifying
their type
            case 2
                row = temp_indc_series*freq_range+i;
                Sp=[Sp S_indc_series(row,:)];
                temp_indc_series=temp_indc_series+1;
            case 3
                row = temp_indc_shunt*freq_range+i;
                Sp=[Sp S_indc_shunt(row,:)];
                temp_indc_shunt=temp_indc_shunt+1;
            case 4
                row = temp_cap_series*freq_range+i;
                Sp=[Sp S_cap_series(row,:)];
                temp_cap_series=temp_cap_series+1;

            case 5
                row = temp_cap_shunt*freq_range+i;
                Sp=[Sp S_cap_shunt(row,:)];
                temp_cap_shunt=temp_cap_shunt+1;

            case 6
                row = temp_coupled*freq_range+i;
                Sp=[Sp S_coupled(row,:)];
                temp_coupled=temp_coupled+1;

            case 7
                row = temp_tee*freq_range+i;
                Sp=[Sp S_tee(row,:)];
                temp_tee=temp_tee+1;

            case 8
                row = temp_line*freq_range+i;
                Sp=[Sp S_line(row,:)];
                temp_line=temp_line+1;
        end
    end
    Sp_all = [Sp_all ;Sp];
end

```

```

        fprintf(fnet, ' %g', port(i));    % put user input value
end
end
fclose(fnet);

%%% Function to read input data file for network topology and set cont_matrix
%%% based on values for elements ports numbers

function [cont_matrix] = netlist_read(Ep)
fnet1 = fopen('Netlist_data');          % open file that contain input data
netlist_old=fscanf(fnet1, ' %g');       % reads stored data for input
fclose(fnet1);

netlist=[];
    for j= 1:24                          % 4 port places for each element
        netlist=[netlist netlist_old(j,1)]; % arrange input data properly to be
                                                % used in the function
    end

temp=1;
netlist_matrix=[];
for i=1:6                                % up to 6 elements in the network
    k=netlist(temp:2+temp);              % take one row of input data at a time
    t=k>0;                               % consider only non-negative values
    temp=temp+4;
    if(sum(t))>0

        % set matrix that contains user input values only
        netlist_matrix=[netlist_matrix; k];
    end
end

element_no=size(netlist_matrix,1);
cont_matrix=zeros(element_no);
for i=1:element_no
    for j=i+1:element_no
        % check for connection
        x=test_connection(netlist_matrix(i,:),netlist_matrix(j,:));
    if (x)
        cont_matrix(i,j)=1;              % set cont_matrix based on connection
        cont_matrix(j,i)=1;
    end
end
end

%%% Function to test connection between ports based on equality in ports
%%% numbers

function equ_element= test_connection(A,B)
equ_element=0;
for i=1:4
    for j=1:4
        equ_element=isequal(A(i),B(j))+ equ_element; % save any equality found
    end
end
end

```

```

%%% Function to set the Connection Matrix for the network based on
%%% previously determined cont_matrix and number of ports of each element
function topo = topo (Np,con_matrix)      % con_matrix describes how elements
                                         % are connected
                                         % Np describe the number of ports of each element

n = size(Np,2);
topo=zeros(n*2+2);

for i=1:n
    if sum(con_matrix(i,:)) < Np(i)
        for k=2:2*n-1      % loop valid for the interconnected ports
            if mod(k,2)==0    % condition depends whether wave number is even or odd
                topo(k,k+1)=1;
            else
                topo(k,k-1)=1;
            end
        end
    else
        % valid for the external input and output ports
        topo(1,2*n+1) =1;
        topo(2*n+1,1) =1;
        topo(2*n,2*n+2) =1;
        topo(2*n+2,2*n) =1;
    end
end
end

```

➤ **Subroutines to Calculate Scattering Parameters for the overall Network based on previously determined Scattering Matrix and Connection Matrix:**

```

%%% Function that determines and displays the Scattering Parameters for the
%%% overall network based on the previously determined Scattering and
%%% Connection Matrices

```

```

function [S]=Cal_S(topology,Smatrix,f1,f2,incrt,Ep)
S=[];
n=1;
n=sum(Ep);
freq_range = size((f1:incrt:f2),2);
k=f1;
for i=1:freq_range      % for the whole range of frequency determined

    %%% determine the Connection_Scattering Matrix
    %%% determine the reflected and incident waves vectors
    [A1, A2 ,B1 ,B2]= W(topology, Smatrix(1:sum(Ep),m:n)) ;

    %%% determine the scattering parameters for specified frequency
    Sf = Calc(A1 ,A2 ,B1, B2);
    m = m+ sum(Ep);
    n= n + sum(Ep);
    S = [S; k Sf];
    k = k+incrt;
end
end

```

```

fprintf (' Frequency \t \t \t \t S11 \t \t \t \t S12 \t \t \t S21 \t \t \t.
\tS22\n');
disp (S)

%%% Function to determine the connection-scattering matrix and incident
%%% and reflected wave vectors

function [A1,A2,B1,B2]=W(gama,S_matrix) % Function returns vectors A and B
                                         %(incident and reflected waves)

n=size(S_matrix,1);
m_matrix=zeros(n+2); % m_matrix is the modified S-matrix with the
                    % appropriate size
m_matrix(1:n,1:n)=S_matrix; % Substituting S-matrix into the modified one
Ibr=gama-m_matrix; % Subtract the S-matrix from the gamma matrix
                    %(W matrix)

C1=zeros(size(gama,1),1);
C2=zeros(size(gama,1),1);
C1(n+1)=1; % Excitation to determine S11 and S21
C1(n+2)=0;
C2(n+1)=0; % Excitation to determine S12 and S22
C2(n+2)=1;
A1=inv(Ibr)*C1; % Calculate the inverse of W matrix.
                % (Determine the incident waves to calculate S11 and S21)
B1=gama*A1; % Determine the reflected waves to calculate S11 and S21
A2=inv(Ibr)*C2; % Determine the incident waves to calculate S12 and S22
B2=gama*A2; % Determine the reflected waves to calculate S12 and S22

%%% Function to calculate the required Scattering parameters from obtained
%%% incident and reflected waves

function [S]=Calc(A1,A2,B1,B2);
a1=A1(1);
b1=B1(1);
aex=A1(end-2); % Incident wave at the external output port
bex=B1(end-2); % Reflected wave at the external output port

S11=b1/a1; % Calculate S11
RE_S11 = real(S11);
IM_S11 = imag(S11);
freflect = fopen('Reflect_data','a'); % open file and write new data
                                         % based on user input
fprintf(freflect,'\n %g %g',RE_S11,IM_S11);
fclose (freflect);

S21=bex/a1; % calculate S21

a1=A2(1);
o1=B2(1);
aex=A2(end-2); % Incident wave at the external output port
oex=B2(end-2); % Reflected wave at the external output port

S12=b1/aex; % Calculate S12
S22=bex/aex; % Calculate S22
S =[S11 S12 S21 S22];

```

➤ Subroutine to Plot Smith Chart from Calculated S11 for the Network

```
%%% Function to determine vector Zl (load impedances) from vector S11
%%% (reflection coefficients)

function S_Zl(f1,f2,incrt,Z0)
Zl=[];
ref_coef=[];
ref_num=[];
freq_range = size((f1:incrt:f2),2);
freflect1= fopen ('Reflect_data');

%%% Read the real and imaginary parts of S11 calculated earlier
for j=1:freq_range
reflect1=fscanf(freflect1, '\n %g %g', [1 2]);
ref_num=[ref_num; reflect1];
end

%%% Form S11 into complex numbers and put them in vector form
for n=1:freq_range
ref_complex=ref_num (n,1)+ref_num(n,2)*i;
ref_coef= [ref_coef ref_complex];
end

%%% Determine vector Zl from vector S11
for m=1:size(ref_coef,2)
Zl(m) = -Z0*(ref_coef(m)+1)/(ref_coef(m)-1);
end

smith (Zl,Z0)
fclose (freflect1);

%%% SMITH (Zl,Zo) draws a Smith chart of the Complex impedances in Zl,
%%% normalized to the real valued characteristic impedance Zo.
%%% Originally based on some Matlab code by Antony-Dean McKechnie & Neville
%%% Wilken in their final year project of their BSc(Eng) at Wits.
%%% But rather heavily rewritten by Alan Robert Clark, PrEng.
% Dept Elect. Eng
% P.O.Wits
% 2050 South Africa
% clark@odie.wits.ee.ac.za
% 15 December 1992.

function smith (Zl, Zo);
% Set up Constant Resistance Circles;
% and Constant reactance circles -- but crop at Outer Circle.
r = [1 0.826 0.665 0.500 0.334 0.162];
axis([-1 1 -1 1]);
Re = [1,2,5,5];
Im = [0.2 0.5 1 2];
for i = 1:4
p = (0:(Re(i)/100):Re(i))';
q = p-p+Im(i);
Icircl = p+j*q;
Icirc2 = p-j*q;
rho1=(Icircl-1)./(Icircl+1);
rho2=(Icirc2-1)./(Icirc2+1);
```

```

    rhomag = abs(rho1);
    Circm(:,i)= rhomag;
    Circp = angle(rho1);
    Circp1(:,i) = Circp;
    Circp = angle(rho2);
    Circp2(:,i) = Circp;
end,
inc = 2*pi/100;
theta = 0:inc:pi;
phi = 0:2*inc:2*pi;
RV = phi-phi+0.329;
for i=1:6
    R1 = 2*r(:,i)*sin(theta+pi/2);
    y1 = R1.*sin(theta);
    x1 = R1.*cos(theta);
    y2 = y1;
    x2 = x1+(1-2*r(i));
    R = sqrt(x2.^2+y2.^2);
    R2(:,i) = R';
    thetal = atan((y2./x2));
for p=1:length(x2) % Crop at 1
    if x2(p) < 0, thetal(p)=thetal(p)-pi; end;
end,
    theta2(:,i) = thetal';
end
hold on
polar(theta2,R2);polar(Circp1,Circm,'--b'); polar( Circp2,Circm);
polar(phi,RV);
% Fill in the Real axis
x=[-pi 0];
y=[1 1];
polar(x,y,'w:');
text(-pi,1.1,'0');
text(-pi,0.66,'.2');
text(-pi,0.33,'.5');
text(0,0,'1');
text(0,0.33,'2');
text(0,0.66,'5');
text(0,1.05,'inf');
text(5.2*pi/6,1.1,'.2');
text(-5.2*pi/6,1.15,'.2');
text(2.2*pi/3,1.1,'.5');
text(-2.2*pi/3,1.15,'.5');
text(pi/2,1.05,'1');
text(-pi/2,1.1,'1');
text(pi/3.2,1.05,'2');
text(-pi/3.2,1.1,'2');
% plot the impedances, -- first normalize
rho=(Zl-Zo)./(Zl+Zo);
rhomag = abs(rho);
rhoph = angle(rho);
polar(rhoph,rhomag,'k*');
hold off % return to usual state
function [rads] = angle (comp_val)
rads = atan2 (imag(comp_val),real(comp_val));

```

➤ Graphical User Interface (GUI) Matlab Files:

1. Series Inductor GUI:

```
% INDUCTOR_SERIES_GUI M-file for inductor_series_gui.fig
function varargout = inductor_series_gui(varargin)
% Begin initialization code
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @inductor_series_gui_OpeningFcn, ...
                  'gui_OutputFcn',  @inductor_series_gui_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code
% --- Executes just before inductor_series_gui is made visible.
function inductor_series_gui_OpeningFcn(hObject, eventdata, handles,
varargin)
% Choose default command line output for inductor_series_gui
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% --- Outputs from this function are returned to the command line.
function varargout = inductor_series_gui_OutputFcn(hObject, eventdata,
handles)
% Get default command line output from handles structure
varargout{1} = handles.output;
function edit1_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in confirm_btn.
function confirm_btn_Callback(hObject, eventdata, handles)
cap_value= str2double(get(handles.edit1,'String'));
close
function edit2_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
set(handles.edit2,'String','0.0');
% --- Executes on button press in pushbutton3.
```



```

function pushbutton3_Callback(hObject, eventdata, handles)
indc1= str2double(get(handles.edit2,'String'));
inductor_series_data(indc1);
close

```

2. Shunt Inductor GUI:

```

% INDUCTOR_SHUNT_GUI M-file for inductor_shunt_gui.fig
function varargout = inductor_shunt_gui(varargin)
% Begin initialization code
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn',  @inductor_shunt_gui_OpeningFcn, ...
                  'gui_OutputFcn',   @inductor_shunt_gui_OutputFcn, ...
                  'gui_LayoutFcn',   [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code
% --- Executes just before inductor_shunt_gui is made visible.
function inductor_shunt_gui_OpeningFcn(hObject, eventdata, handles,
varargin)
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% --- Outputs from this function are returned to the command line.
function varargout = inductor_shunt_gui_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;
function edit1_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in confirm_btn.
function confirm_btn_Callback(hObject, eventdata, handles)
cap_value= str2double(get(handles.edit1,'String'));
close
function edit2_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pushbutton1.

```

```

function pushbutton1_Callback(hObject, eventdata, handles)
set (handles.edit2,'String','0.0');
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
indc2= str2double(get(handles.edit2,'String'));
inductor_shunt_data(indc2);
close

```

3. Series Capacitor GUI:

```

% CAPACITOR_SERIES_GUI M-file for capacitor_series_gui.fig
function varargout = capacitor_series_gui(varargin)
% Begin initialization code
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @capacitor_series_gui_OpeningFcn, ...
'gui_OutputFcn',  @capacitor_series_gui_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
[varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
gui_mainfcn(gui_State, varargin{:});
end
% End initialization code
% --- Executes just before capacitor_series_gui is made visible.
function capacitor_series_gui_OpeningFcn(hObject, eventdata, handles,
varargin)
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% --- Outputs from this function are returned to the command line.
function varargout = capacitor_series_gui_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;
function edit1_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in confirm_btn.
function confirm_btn_Callback(hObject, eventdata, handles)
cap_value= str2double(get(handles.edit1,'String'));
close
function edit2_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
set(hObject,'BackgroundColor','white');

```

```

end
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
set(handles.edit2,'String','0.0');
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
cap2= str2double(get(handles.edit2,'String'));
capacitor_series_data(cap2);
close

```

4. Shunt Capacitor GUI:

```

% CAPACITOR_SHUNT_GUI M-file for capacitor_shunt_gui.fig
function varargout = capacitor_shunt_gui(varargin)
% Begin initialization code
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn',  @capacitor_shunt_gui_OpeningFcn, ...
                  'gui_OutputFcn',  @capacitor_shunt_gui_OutputFcn, ...
                  'gui_LayoutFcn',   [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code
% --- Executes just before capacitor_shunt_gui is made visible.
function capacitor_shunt_gui_OpeningFcn(hObject, eventdata, handles,
varargin)
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% --- Outputs from this function are returned to the command line.
function varargout = capacitor_shunt_gui_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;
function edit1_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in confirm_btn.
function confirm_btn_Callback(hObject, eventdata, handles)
cap_value= str2double(get(handles.edit1,'String'));
close
function edit2_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)

```

```

function edit2_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
set(handles.edit2,'String','0.0');
set(handles.edit3,'String','0.0');
set(handles.edit4,'String','0.0');
set(handles.edit5,'String','0.0');
set(handles.edit6,'String','0.0');
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
Width= str2double(get(handles.edit2,'String'));
Height= str2double(get(handles.edit3,'String'));
Length= str2double(get(handles.edit4,'String'));
Seperation= str2double(get(handles.edit5,'String'));
D_constant= str2double(get(handles.edit6,'String'));
coupled_data(Width,Height,Length,Seperation,D_constant);
close
function edit3_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit4_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit5_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit6_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

```

6. Microstrip Tee Junction GUI:

```
TEE_GUI M-file for tee_gui.fig
function varargout = tee_gui(varargin)
% Begin initialization code
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn', @tee_gui_OpeningFcn, ...
                  'gui_OutputFcn',  @tee_gui_OutputFcn, ...
                  'gui_LayoutFcn',   [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code
% --- Executes just before tee_gui is made visible.
function tee_gui_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% --- Outputs from this function are returned to the command line.
function varargout = tee_gui_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;
function edit1_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in confirm btn.
function confirm_btn_Callback(hObject, eventdata, handles)
cap_value= str2double(get(handles.edit1,'String'));
close
function edit2_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
set (handles.edit2,'String','0.0');
set (handles.edit3,'String','0.0');
set (handles.edit4,'String','0.0');
set (handles.edit6,'String','0.0');
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
fidth1= str2double(get(handles.edit2,'String'));
```

```

Width2= str2double(get(handles.edit3,'String'));
Height= str2double(get(handles.edit4,'String'));
D_constant= str2double(get(handles.edit6,'String'));
tee_junc_data(Width1,Width2,Height,D_constant);
close
function edit3_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit4_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit5_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit6_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

7. Microstrip Open-End Transmission Line GUI:

```

% OPEN_GUI M-file for open_gui.fig
function varargout = open_gui(varargin)
% Begin initialization code
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @open_gui_OpeningFcn, ...
                  'gui_OutputFcn',  @open_gui_OutputFcn, ...
                  'gui_LayerFcn',   [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code

```

```

% --- Executes just before open_gui is made visible.
function open_gui_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% --- Outputs from this function are returned to the command line.
function varargout = open_gui_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;
function edit1_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in confirm_btn.
function confirm_btn_Callback(hObject, eventdata, handles)
cap_value= str2double(get(handles.edit1,'String'));
close
function edit2_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
set (handles.edit2,'String','0.0');
set (handles.edit3,'String','0.0');
set (handles.edit4,'String','0.0');
set (handles.edit6,'String','0.0');
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
Width= str2double(get(handles.edit2,'String'));
Height= str2double(get(handles.edit3,'String'));
Length= str2double(get(handles.edit4,'String'));
D_constant= str2double(get(handles.edit6,'String'));
open_end_data(Width,Height,Length,D_constant);
close
function edit3_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit4_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit5_Callback(hObject, eventdata, handles)

```

```

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit6_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

8. Network Topology GUI:

```

% TOPO_GUI M-file for topo_gui.fig
function varargout = topo_gui(varargin)
% Begin initialization code
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @topo_gui_OpeningFcn, ...
                  'gui_OutputFcn',  @topo_gui_OutputFcn, ...
                  'gui_LayerFcn',   [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code
% --- Executes just before topo_gui is made visible.
function topo_gui_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% --- Outputs from this function are returned to the command line.
function varargout = topo_gui_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function edit3_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit4_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```



```

end
function edit5_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit6_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit7_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit8_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit9_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit10_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit11_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit12_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function edit13_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit14_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit15_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit16_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit16_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit17_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit17_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit18_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit18_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit19_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit19_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit20_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit20_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pushbutton1.

```

```

function pushbutton1_Callback(hObject, eventdata, handles)
set (handles.edit3,'String',' ');
set (handles.edit4,'String',' ');
set (handles.edit5,'String',' ');
set (handles.edit6,'String',' ');
set (handles.edit7,'String',' ');
set (handles.edit8,'String',' ');
set (handles.edit9,'String',' ');
set (handles.edit10,'String',' ');
set (handles.edit11,'String',' ');
set (handles.edit12,'String',' ');
set (handles.edit13,'String',' ');
set (handles.edit14,'String',' ');
set (handles.edit15,'String',' ');
set (handles.edit16,'String',' ');
set (handles.edit17,'String',' ');
set (handles.edit18,'String',' ');
set (handles.edit19,'String',' ');
set (handles.edit20,'String',' ');
set (handles.edit24,'String',' ');
set (handles.edit25,'String',' ');
set (handles.edit26,'String',' ');
set (handles.edit27,'String',' ');
set (handles.edit28,'String',' ');
set (handles.edit29,'String',' ');
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
p1= str2double(get(handles.edit3,'String'));
p2= str2double(get(handles.edit4,'String'));
p3= str2double(get(handles.edit5,'String'));
p4= str2double(get(handles.edit24,'String'));
p5= str2double(get(handles.edit6,'String'));
p6= str2double(get(handles.edit7,'String'));
p7= str2double(get(handles.edit8,'String'));
p8= str2double(get(handles.edit25,'String'));
p9= str2double(get(handles.edit9,'String'));
p10= str2double(get(handles.edit10,'String'));
p11= str2double(get(handles.edit11,'String'));
p12= str2double(get(handles.edit26,'String'));
p13= str2double(get(handles.edit12,'String'));
p14= str2double(get(handles.edit13,'String'));
p15= str2double(get(handles.edit14,'String'));
p16= str2double(get(handles.edit27,'String'));
p17= str2double(get(handles.edit15,'String'));
p18= str2double(get(handles.edit16,'String'));
p19= str2double(get(handles.edit17,'String'));
p20= str2double(get(handles.edit28,'String'));
p21= str2double(get(handles.edit18,'String'));
p22= str2double(get(handles.edit19,'String'));
p23= str2double(get(handles.edit20,'String'));
p24= str2double(get(handles.edit29,'String'));
port =[p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14 p15 p16 p17 p18 p19 p20
p21 p22 p23 p24];
setlist_data (port);
close
function edit24_Callback(hObject, eventdata, handles)

```

```

% --- Executes during object creation, after setting all properties.
function edit24_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit25_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit25_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit26_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit26_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit27_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit27_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit28_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit28_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit29_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit29_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit30_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit30_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit31_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit31_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit32_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.

```

```

function edit32_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit33_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit33_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit34_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit34_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit35_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit35_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit36_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit36_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit37_Callback(hObject, eventdata, handles)
% --- Executes during object creation, after setting all properties.
function edit37_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end

```

9. Microwave Circuit Analysis GUI:

```

% MICROWAVE_GUI M-file for microwave_gui.fig
function varargout = microwave_gui(varargin)
% Begin initialization code
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn',  @microwave_gui_OpeningFcn, ...
                  'gui_OutputFcn',  @microwave_gui_OutputFcn, ...
                  'gui_LayerFcn',    [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargin

```

```

    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code
% --- Executes just before microwave_gui is made visible.
function microwave_gui_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% --- Outputs from this function are returned to the command line.
function varargout = microwave_gui_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
menu1 = get(hObject, 'Value');
switch menu1
    case 2
        inductor_series_gui;
    case 3
        inductor_shunt_gui;
    case 4
        capacitor_series_gui;
    case 5
        capacitor_shunt_gui;
    case 6
        coupled_gui;
    case 7
        tee_gui;
    case 8
        open_gui;
end
% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end
% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
menu2 = get(hObject, 'Value');
switch menu2
    case 2
        inductor_series_gui;
    case 3
        inductor_shunt_gui;
    case 4
        capacitor_series_gui;
    case 5
        capacitor_shunt_gui;
    case 6
        coupled_gui;
    case 7
        tee_gui;
    case 8
        open_gui;
end
end

```

```

% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
menu3 = get(hObject,'Value');
switch menu3
    case 2
        inductor_series_gui;
    case 3
        inductor_shunt_gui;
    case 4
        capacitor_series_gui;
    case 5
        capacitor_shunt_gui;
    case 6
        coupled_gui;
    case 7
        tee_gui;
    case 8
        open_gui;
end
% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on selection change in popupmenu4.
function popupmenu4_Callback(hObject, eventdata, handles)
menu4 = get(hObject,'Value');
switch menu4
    case 2
        inductor_series_gui;
    case 3
        inductor_shunt_gui;
    case 4
        capacitor_series_gui;
    case 5
        capacitor_shunt_gui;
    case 6
        coupled_gui;
    case 7
        tee_gui;
    case 8
        open_gui;
end
% --- Executes during object creation, after setting all properties.
function popupmenu4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pushbutton1.

```

```

function pushbutton1_Callback(hObject, eventdata, handles)
function edit3_Callback(hObject, eventdata, handles)
incrt= str2double(get(handles.edit3,'String'));
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit4_Callback(hObject, eventdata, handles)
f1= str2double(get(handles.edit4,'String'));
% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit2_Callback(hObject, eventdata, handles)
f2= str2double(get(handles.edit2,'String'));
% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit5_Callback(hObject, eventdata, handles)
Z0= str2double(get(handles.edit5,'String'));
% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
topo_gui;
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
    menu2 = get(handles.popupmenu2, 'Value');
    menu3 = get(handles.popupmenu3, 'Value');
    menu4 = get(handles.popupmenu4, 'Value');
    menu5 = get(handles.popupmenu6, 'Value');
    menu6 = get(handles.popupmenu7, 'Value');
    menus = [menu1 menu2 menu3 menu4 menu5 menu6];
    f1=str2double(get(handles.edit4,'String'));
    f2=str2double(get(handles.edit2,'String'));
    incrt=str2double(get(handles.edit3,'String'));
    Z0=str2double(get(handles.edit5,'String'));
    Smatrix=individual_S (menus,f1,f2,incrt,Z0);
    Ep=elements (menus);
    connection=netlist_read(Ep);
    topology = topo (Ep,connection);
    freflect = fopen('Reflect_data','w'); % open file and clear stored data
    fclose(freflect);
    Cal_S(topology,Smatrix,f1,f2,incrt,Ep);
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)

```



```

f1=str2double(get(handles.edit4,'String'));
f2=str2double(get(handles.edit2,'String'));
incrt=str2double(get(handles.edit3,'String'));
Z0=str2double(get(handles.edit5,'String'));
figure
S_Z1(f1,f2,incrt,Z0)
% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
initial_all;
set(handles.popupmenu1,'Value',1);
set(handles.popupmenu2,'Value',1);
set(handles.popupmenu3,'Value',1);
set(handles.popupmenu4,'Value',1);
set(handles.popupmenu6,'Value',1);
set(handles.popupmenu7,'Value',1);
set(handles.edit4,'String','0.0');
set(handles.edit2,'String','0.0');
set(handles.edit3,'String','0.0');
set(handles.edit5,'String','0.0');
% --- Executes on selection change in popupmenu6.
function popupmenu6_Callback(hObject, eventdata, handles)
menu5 = get(hObject,'Value');
switch menu5
    case 2
        inductor_series_gui;
    case 3
        inductor_shunt_gui;
    case 4
        capacitor_series_gui;
    case 5
        capacitor_shunt_gui;
    case 6
        coupled_gui;
    case 7
        tee_gui;
    case 8
        open_gui;
end
% --- Executes during object creation, after setting all properties.
function popupmenu6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on selection change in popupmenu7.
function popupmenu7_Callback(hObject, eventdata, handles)
menu6 = get(hObject,'Value');
switch menu6
    case 2
        inductor_series_gui;
    case 3
        inductor_shunt_gui;
    case 4
        capacitor_series_gui;
    case 5
        capacitor_shunt_gui;
    case 6

```

```
        coupled_gui;
    case 7
        tee_gui;
    case 8
        open_gui;
end
% --- Executes during object creation, after setting all properties.
function popumenu7_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```