An Object-Oriented Co-Simulation Framework for Interconnecting PSpice Circuits with Time-Domain Electromagnetic Field Simulators

by

Yingying Lu

B.Sc, McMaster University, 2006

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Supervisory Committee

An Object-Oriented Co-Simulation Framework for Interconnecting PSpice Circuits with Time-Domain Electromagnetic Field Simulators

by

Yingying Lu

B.Sc, McMaster University, 2006

Supervisory Committee

Dr. Poman So, (Department of Electrical and Computer Engineering)
Supervisor

Dr. Wolfgang Hoefer, (Department of Electrical and Computer Engineering)
Co-Supervisor or Departmental Member

Dr. Yvonne Coady, (Department of Computer Science)
Outside Member

Abstract

Supervisory Committee

Dr. Poman So, (Department of Electrical and Computer Engineering)
Supervisor

Dr. Wolfgang Hoefer, (Department of Electrical and Computer Engineering)
Co-Supervisor

Dr. Yvonne Coady, (Department of Computer Science)
Outside Member

The proliferation of wireless communication has driven up the demand for electromagnetic and circuit co-simulations. Electromagnetic field solvers such as CST Microwave Studio and Flomerics Microstripes have proprietary links to Agilent ADS and AWR Microwave Office, respectively. However, popular electronics circuit solvers such as Cadence PSpice cannot handle co-simulation of circuits and fields effectively. In this thesis, a co-simulation framework is developed for interfacing PSpice with electromagnetic field simulators. Algorithms for direct one-port and two-port connections between Transmission Line Matrix (TLM) models of electromagnetic structures and PSpice, and convolution algorithms for connecting TLM structures with PSpice will be presented. The latter algorithm is more efficient than the former, and it employs the pre-computed impulse response (Johns response) of the structures. Convolution algorithms for time domain diakoptics of TLM structures using one-port and two-port Johns Matrices for TEM waveguide and TE_{10} waveguide applications will also be presented.

Table of Contents

List of Tables

List of Figures

List of Symbols

Symbols

c	speed of light	$[C]$	connecting matrix
f	frequency	$[G]$	Johns Matrix
λ	wavelength	v_l	wave velocity on the link lines
t	time	v_n	wave velocity on the TLM network or in the medium
k	integer designating the number of time steps in a simulation	Z_l	characteristic impedance of TLM link line
ε	absolute permittivity	Y_l	characteristic admittance of TLM link line
ε_o	permittivity of free space	Z_o	characteristic impedance of free space
ε_r	relative permittivity	Y_o	characteristic admittance of free space
μ	absolute permeability	V	voltage
μ_o	permeability of free space	C	capacitance
μ_r	relative permeability	L	inductance
σ_e	equivalent electric conductivity	V_{TLM}	Thévenin equivalent voltage source for the TLM mesh
σ_m	equivalent magnetic conductivity	Z_{TLM}	Thévenin equivalent impedance for the TLM mesh
Δt	time step	E	electric field intensity
Δl	mesh parameter or cell size	H	magnetic field intensity
$[S]$	scattering matrix		

Common Abbreviations

2D	Two-Dimensional
3D	Three-Dimensional
GSCN	Generalized Symmetrical Condensed Node
MEFiSTo	Multi-Purpose Electromagnetic Field Simulation Tool
PSpice	A PC version of SPICE (from MicroSim Corporation)
RF	Radio Frequency
SCN	Symmetrical Condensed Node
S-parameters	Scattering parameters (Complex ratio of power waves incident and reflected at ports of microwave networks)
SPICE	Simulation Program with Integrated Circuits Emphasis
TE	Transverse Electric
TEM	Transverse Electromagnetic
TLM	Transmission Line Matrix
TM	Transverse Magnetic
TE-to-y	Transverse Electric; magnetic field vector is in the y-direction, electric field vector is in the x-z plane
TM-to-y	Transverse Magnetic; electric field vector is in the y-direction, magnetic field vector is in the x-z plane

List of Classes

Class Array3D	This is a generic three-dimensional array class. Objects of this class can be treated as two- and one-dimensional arrays. Array elements can be accessed using the () operator. It isot necessary to use this class for one-dimensional array if thee bulit-in C/C++ array meet your need.
Class TLM	This class handles electromagnetic field simulations in the Transverse Electromagnetic (TEM) mode.
Class TLM_TE	This class handles electromagnetic field simulations in the Transverse Electric (TE) mode.
Class PSpice_cir	This class performs the calculations involved with circuit simulator PSpice.
Class TLM_Pspice	This class makes the direct connection between TLM structures and PSpice circuits.
Class Conv_Pspice	This class makes the connection between TLM structures and PSpice circuits by convolution

Acknowledgments

I would like to express my sincere gratitude and appreciation to my supervisors Dr. Wolfgang Hoefer and Dr. Poman So for providing me the valuable opportunity to work in the research area of Computational Electromagnetics, for their mentorship and informative guidance, and for their understanding and untiring support. I would also like to thank Dr. Jens Bornemann for inspiring discussions in the field of antenna design. Additional appreciation is expressed to Dr. Yvonne Coady for her insightful suggestions and comments.

I am grateful to Donna Shannon, all my friends and colleagues at the University of Victoria. Their support, insightful discussions and friendship made the past two years memorable.

Finally, I would like to express my deepest appreciation to my parents and my husband for their encouragement and moral support throughout my studies.

Dedication

To my parents

Chapter 1

Introduction

1.1 Background

The Transmission Line Matrix Modeling Method (TLM) is a time-domain numerical technique for solving general wave problems. In the TLM model, the field space is filled with a network of transmission lines. Voltage pulses travel from one node to another in the network in a fixed time step [1]. The method was first proposed in the early 1970's by P.B. Johns and his coworkers based on Huygens' model of wave propagation [2, 3]. According to Huygens' model, a wavefront can be regarded as a number of secondary radiators that generate spherical wavelets. Those wavelets form a new wavefront which generates new spherical wavelets [2]. This process occurs repeatedly.

Time domain diakoptics, on the other hand is a very important technique which was first introduced by Kron [4] . Diakoptics means breaking down a large network into subnetworks for individual analysis and then reconnecting those subnetworks together [5, 6]. Diakoptics is applied in this research work. The main idea is to break down a big TLM structure into several small substructures for individual analyses. Each TLM substructure is characterized by an impulse response called a "Johns Matrix". The overall response of the complete TLM structure is obtained using convolution. In addition to connecting TLM substructures together, this diakoptics technique can also be applied to connect SPICE circuits to TLM structures. SPICE circuits and TLM structures can be analyzed independently to get sub-responses. The overall response of the whole structure, which contains both SPICE circuits and TLM structures, is obtained by convolution of sub-responses.

1.2 Motivation

It is known that certain electronic systems contain components that are best analyzed by electromagnetic (EM) fields, while other components are best analyzed by circuits. To solve such a combined system, we need to establish a connection between such components to achieve a co-simulation framework. PSpice is a circuit simulation software package, which is very popular by electronic designers; MEFiSTo is an electromagnetic field simulator based on the Transmission Line Matrix (TLM) method. A co-simulation framework is needed to connect PSpice with field solvers in order to solve complex systems.

Secondly, a large structure can be divided into smaller substructures for individual analysis and then recombined to obtain the overall response. In the frequency domain, the connection of substructures is well established using the S-matrix or ABCD matrix. My contribution is to do the connection in the time domain. This divide and conquer process is called diakoptics. There are two reasons for using diakoptics: first, it is easier to solve small problems, since small problems are less complex and consume less memory and CPU time. Second, when a substructure is altered, it is not necessary to solve the whole structure again; we only need to solve the substructure that needs to be changed.

The connection algorithms for field and circuit solvers or substructures for field simulations, involve convolution at the interface in the time domain.

1.3 Research Objectives

The objectives of this research work are as follows:

1) To develop convolution algorithms for time domain diakoptics of TLM structures.
2) To design a co-simulation framework for connecting Cadence PSpice with time domain electromagnetic field simulators.

1.4 Original Contributions

This thesis features the following original contributions in the area of time-domain modeling of electromagnetic structures.

1) A novel object-oriented framework for connecting the TLM simulator MEFiSTo with PSpice by convolution in the time domain has been developed and implemented. (Chapter 3)
2) A novel TLM convolution algorithm for cascading two-port electromagnetic structures characterized by a 2×2 Johns Matrix has been developed and implemented. (Chapters 4 and 6)
3) A novel convolution algorithm for connecting two-port electromagnetic structures modeled by TLM with the commercial circuit simulator PSpice has been developed and implemented. (Chapters 5 and 6)

All algorithms have been validated against full-wave simulations performed with MEFiSTo.

1.5 Thesis Structure

In Chapter 2, an overview of TLM is given. This chapter provides the background information on applying diakoptics to two- and three-dimensional TLM methods.

In Chapter 3, the design and implementation of a co-simulation framework for interconnecting PSpice circuits with a time-domain electromagnetic field simulator are presented.

In Chapter 4, convolution algorithms for TLM structures using the one-port Johns Matrix and the two-port Johns Matrix are proposed. A concatenation of three Transverse Electromagnetic (TEM) waveguides by convolution and a concatenation of three TE_{10} waveguides by convolution are also presented.

In Chapter 5, a co-simulation framework for direct one-port and two-port connection between PSpice and TLM, and connection of PSpice with TLM using convolution are proposed.

In Chapter 6, two validation examples of the co-simulation algorithms are presented, namely single stub tuning in a TEM waveguide and an inductive iris in a TE_{10} waveguide.

Chapter 7, the final chapter of this thesis, concludes the research findings and proposes future research and development activities for generalizing the implementation and transfer of the technology to the CAD industry.

Chapter 2

Overview of the Transmission Line Matrix Method (TLM)

This chapter gives an overview of four commonly used TLM node structures, namely the two-dimensional Shunt and Series Nodes as well as the three-dimensional Symmetrical Condensed Node (SCN) and General Symmetrical Condensed Node (GSCN).

2.1 Two-Dimensional TLM Method

The two-dimensional (2D) TLM Method can be used to model electromagnetic fields that vary along two spatial dimensions only. This includes fields in three-dimensional (3D) space where all three field components vary along two spatial dimensions and have no variation in the third spatial dimension [2]. There are two kinds of 2D TLM networks: one is based on the Shunt Node and the other on the Series Node. As the names imply, a shunt Node TLM is formed when transmission lines are connected in shunt, while a Series Node TLM is formed when transmission lines are connected in series.

2.1.1 Shunt Node TLM

Figure 2-1 shows the 2D TLM shunt node [7, 8] which is the basic building block, or unit cell, of the 2D TLM shunt mesh, and its equivalent lumped element model [2]. The distance between two adjacent nodes Δl is called mesh parameter or cell size. Figure 2-2 shows a schematic of a basic 2D TLM network. A node is at the intersection of four link lines and is the point at which scattering occurs. A cell contains a node and four link lines, each $0.5\Delta l$ in length. The cell size is Δl. TLM boundaries are defined at the cell boundaries. The 2D TLM

network may contain square or rectangular cells. In the latter case the cell size in the z- and x-directions are different. In this thesis, only 2D TLM networks with square cells are discussed.

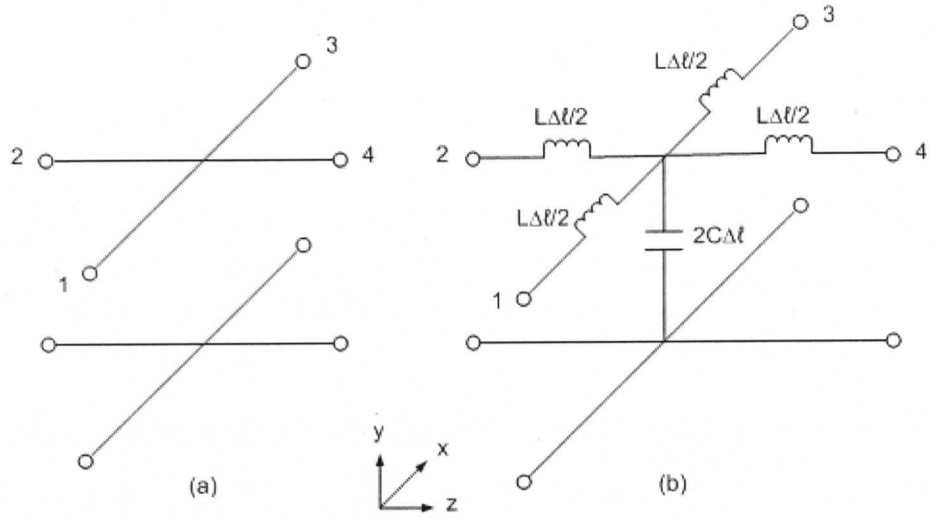


Figure 2-1 (a) Shunt node building block of the two-dimensional square TLM network. (b) Equivalent lumped element model.

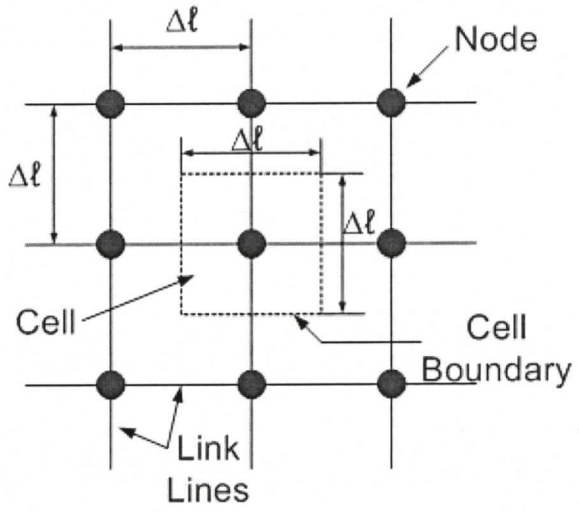


Figure 2-2 Illustration of a basic 2D TLM network.

In a typical time-domain TLM simulation, short voltage impulses are injected into the transmission line network. These impulses travel along the

transmission lines towards the nodes. As the impulses arrived at the nodes, they are scattered and travel towards the cell interface. This process is depicted in Figure 2-3. At the beginning of the simulation, all voltage impulses in the mesh are usually set to zero. A unit voltage impulse, with unit energy, is incident on the cell boundary at $t_0 = k\Delta t$ (where k is the number of time-steps and Δt is the time taken for an impulse to travel between nodes). When the unit voltage impulse reaches the node center at time $t = (k+1/2)\Delta t$, scattering occurs. A reflected impulse of $-1/2V$ is scattered backward, along the line of incidence, and three transmitted impulses of $+1/2V$ are scattered forward into the three outgoing transmission lines as shown in Figure 2-3(a). The four scattered voltage impulses travel along the transmission lines and arrive at the cell boundary at time $t = (k+1)\Delta t$. The scattered voltage impulses are exchanged between neighbouring nodes and become incident impulses connecting to neighbouring nodes at $t = (k+1)\Delta t$ as shown in Figure 2-3(b). They arrive at the center of adjacent nodes at $t = (k+3/2)\Delta t$ [8]. This process is described by the connection matrix given in Equation (2.4). When the medium electromagnetic wave travels through changes suddenly at a interface, reflection process occurs at the interface as shown in Figure 2-3(c). The sequence of scattering and transmission or reflection is repeated recursively until the end of computation.

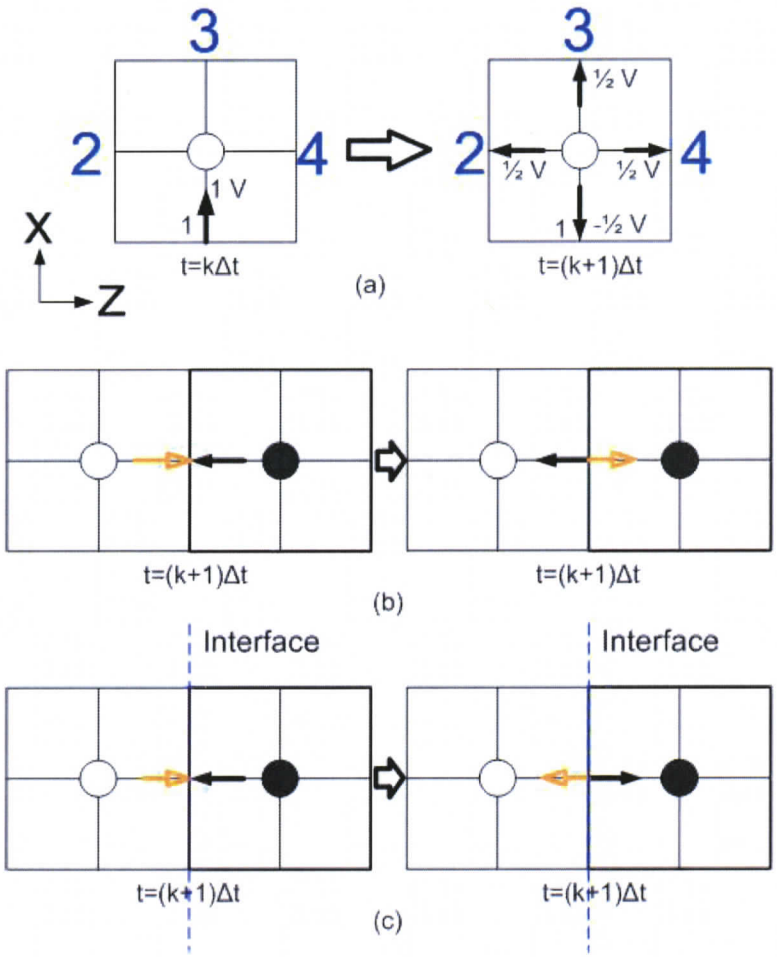


Figure 2-3 TLM algorithm for the 2D shunt node: (a) Scattering (b) Connecting (c)Reflecting.

Incident voltage impulses ${}_kV_1^i$, ${}_kV_2^i$, ${}_kV_3^i$, and ${}_kV_4^i$ enter the cell boundary at $t = k\Delta t$. Scattered voltage impulses ${}_{k+1}V_1^r$, ${}_{k+1}V_2^r$, ${}_{k+1}V_3^r$, and ${}_{k+1}V_4^r$ are reflected back to the cell boundary at $t = (k+1)\Delta t$. The reflected voltage impulse in line n at time $t = (k+1)\Delta t$ is given in Equation (2.1):

$$ {}_kV_n^r = V_{node} - {}_kV_n^i \tag{2.1}$$

where the voltage at a node V_{node} is:

$$ V_{node} = (1-\frac{1}{2})({}_kV_1^i + {}_kV_2^i + {}_kV_3^i + {}_kV_4^i) = \frac{1}{2}\sum_{m=1}^{4} {}_kV_m^i \tag{2.2}$$

The above scattering procedure can be formulated using matrix multiplication as shown in Equation (2.3).

$$[{}_{k+1}V^r] = [S][{}_kV^i] \tag{2.3 a}$$

where

$$[S] = \frac{1}{2}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \quad [{}_{k+1}V^r] = {}_{k+1}\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}^r \quad [{}_kV^i] = {}_k\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}^i \tag{2.3 b}$$

The connection matrix $[C]$ is used to relate reflected voltage impulses at $t = k\Delta t$ to voltage impulses incident on the neighbouring nodes at $t = (k+1)\Delta t$, as shown in Equation (2.4) [9].

$$[{}_{k+1}V^i] = [C][{}_{k+1}V^r] \tag{2.4 a}$$

where

$$[C] = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad [{}_{k+1}V^i] = \begin{bmatrix} {}_{k+1}V_1^i(z,x) \\ {}_{k+1}V_2^i(z,x) \\ {}_{k+1}V_3^i(z,x) \\ {}_{k+1}V_4^i(z,x) \end{bmatrix} \quad [{}_{k+1}V^r] = \begin{bmatrix} {}_{k+1}V_1^i(z,x+1) \\ {}_{k+1}V_2^i(z+1,x) \\ {}_{k+1}V_3^i(z,x-1) \\ {}_{k+1}V_4^i(z-1,x) \end{bmatrix}$$

and z and x are shown in Figure 2-3 (a) (2.4 b)

As long as the cell size is much smaller than the wavelength of interest i.e. $\Delta l << \lambda$, each cell can be considered infinitesimal. Voltage and current on the mesh can be related to equivalent field quantities via two sets of differential equations as shown in Equation (2.5) for the TM-to-y case ([2, 10]).

Voltage and current relationship	E and H relationship
$\frac{\partial V_y}{\partial x} = -L\frac{\partial I_x}{\partial t}$	$\frac{\partial E_y}{\partial x} = -\mu_n\frac{\partial H_z}{\partial t}$
$\frac{\partial V_y}{\partial z} = -L\frac{\partial I_z}{\partial t}$	$\frac{\partial E_y}{\partial z} = \mu_n\frac{\partial H_x}{\partial t}$
$\frac{\partial I_z}{\partial z} + \frac{\partial I_x}{\partial x} = -2C\frac{\partial V_y}{\partial t}$	$\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} = \varepsilon_n\frac{\partial E_y}{\partial t}$

$$E_y \equiv V_y;\ \ H_z \equiv I_x;\ \ H_x \equiv -I_z;\ \ \mu_n \equiv L;\ \varepsilon_n \equiv 2C \tag{2.5}$$

where μ_n and ε_n are the permeability and permittivity of the virtual medium modeled by the TLM network, respectively.

Equations (2.6) and (2.7) show that the wave velocity in the link line v_l is $\sqrt{2}$ times the wave velocity on the TLM network or in the medium v_n; while the link line characteristic impedance Z_l is $\sqrt{2}$ times the intrinsic impedance Z_{no} of the TLM network.

$$v_l = \frac{1}{\sqrt{LC}} = \frac{\sqrt{2}}{\sqrt{\varepsilon_n \mu_n}} = v_n \sqrt{2} \tag{2.6}$$

$$Z_l = \sqrt{\frac{L}{C}} = \sqrt{\frac{\mu_n}{\varepsilon_n}}\sqrt{2} = Z_{no}\sqrt{2} \tag{2.7}$$

2.1.2 Series Node TLM

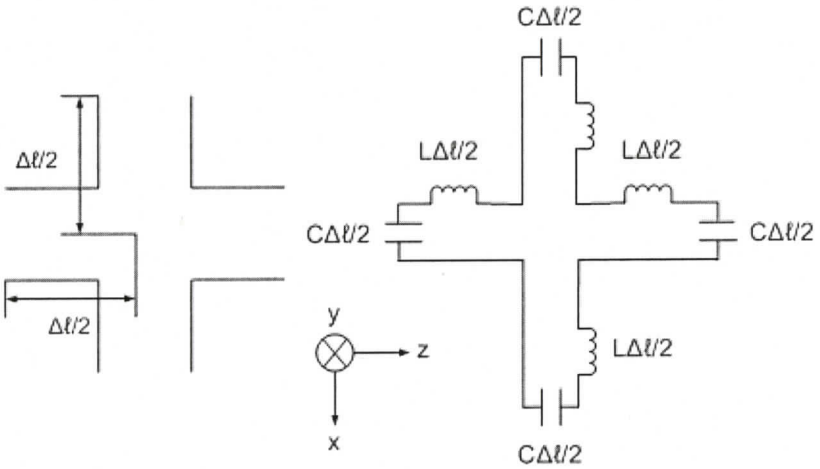


Figure 2-4 Schematic of the 2D series node: (a) Transmission line model (b) Equivalent lumped element model.

The 2D Series Node (as shown in Figure 2-4) connects four transmission lines in series. It has similar characteristics as the 2D Shunt Node. Babinet's principle states that intensity distributions produced by two complementary diffracting screens are identical. Two screens are called complementary when opaque parts of one screen are the transparent parts of the other screen, and the

transparent parts of one screen are the opaque parts of the other screen. According to Babinet's principle, which is based on the dual nature of the electric and the magnetic field, the Series Node algorithm can be obtained from Shunt Node TLM by interchanging electric field with magnetic field, impedance with admittance, and permeability with permittivity of the medium [2]. The resulting transmission line equations and the corresponding 2D field equations are shown in Equation (2.8) for the TE-to-y case:

Voltage and current relationship	E and H relationship
$\frac{\partial I_y}{\partial x} = -C\frac{\partial V_z}{\partial t}$	$\frac{\partial H_y}{\partial x} = -\varepsilon_n\frac{\partial E_z}{\partial t}$
$\frac{\partial I_y}{\partial z} = -C\frac{\partial V_x}{\partial t}$	$\frac{\partial H_y}{\partial z} = \varepsilon_n\frac{\partial E_x}{\partial t}$
$\frac{\partial V_x}{\partial z} + \frac{\partial V_z}{\partial x} = -2L\frac{\partial I_y}{\partial t}$	$\frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x} = -\mu_n\frac{\partial H_y}{\partial t}$

$$H_y \equiv I_y ; E_x \equiv V_x ; E_z \equiv -V_z ;\ \mu_n \equiv 2L ; \varepsilon_n \equiv C \tag{2.8}$$

The relationships between E and H have been derived from Maxwell's equations by setting $\partial/\partial y = 0$, $E_y = H_x = H_z = 0$. As in the Shunt Node case, wave velocity in the network and intrinsic admittance of the network are smaller than their counterpart in the link lines by a factor of $\frac{1}{\sqrt{2}}$. Using the same approach as that described for the 2D shunt node, the scattering procedure for the 2D series node TLM is obtained as shown in Equation (2.9):

$$[{}_{k+1}V^r] = [S][{}_kV^i] \tag{2.9 a}$$

where

$$[S] = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{bmatrix} \qquad [{}_{k+1}V^r] = {}_{k+1}\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}^r \qquad [{}_kV^i] = {}_k\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}^i \tag{2.9 b}$$

2.2 Three-Dimensional TLM Method

2.2.1 Symmetrical Condensed Node TLM

In 1987, Peter B. Johns introduced the 3D Symmetrical Condensed Node (SCN) for the TLM method [1] . The 3D SCN node (Figure 2-5) is in the center of a cubic cell with six branches connected to it. Each branch has two orthogonal polarizations, each corresponding to one pair of uncoupled transmission lines. There are twelve transmission lines in one cell. All the transmission lines have the same characteristic impedances which is equal to the characteristic impedances of free space Z_0 [1]. Voltage pulses (which are labelled as V_1 to V_{12} in Figure 2-5) are associated with polarizations in the cell. Stubs are introduced to model permittivity and permeability, which are equivalent to extra inductances and capacitances added to the node.

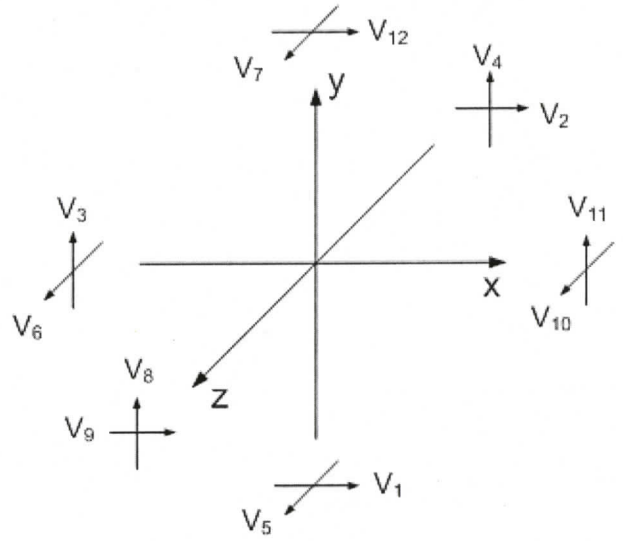


Figure 2-5 The structure of a 3D SCN node for one unit cell.

As in the 2D shunt node TLM, an impulse scattering matrix is used to relate incident pulses to reflected pulses. When incident voltage pulses V^i are incident on the node, twelve reflected pulses V^r are scattered into the twelve ports at the terminals of transmission lines. This relationship is shown in Equation 2.10. By imposing conservation of voltage, current and energy at the node [11], the

12×12 scattering matrix can be derived as shown in Equation 2.11. Both V^i and V^r are vectors with 12 components.

$$ {}_kV^r = S_kV^i \tag{2.10} $$

$$ S=\frac{1}{2}\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & -1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{2.11} $$

As in the 2D nodes, voltage impulses reflected at one node become incident voltage impulses on adjacent nodes. The scattering and connection procedures are repeated until the end of computation.

In addition to the twelve link lines, six extra stubs are needed to represent ε_x, ε_y, ε_z, μ_x, μ_y and μ_z; these stubs are connected to ports 13, 14, 15, 16, 17, 18, respectively. Permittivity stubs are open-circuited, adding capacitance to the node, whereas permeability stubs are short-circuited, adding inductance to the node. The scattering matrix $[S]$ then becomes an 18×18 matrix which is shown in Equation (2.12) [1].

#			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
			x	x	y	y	z	z	z	y	x	z	y	x	x	y	z			
			z	y	z	x	x	y	x	x	y	y	z	z				x	y	z
1	x	z	a	b	d						b	$-d$		c	g					i
2	x	y	b	a				d			c	$-d$		b	g				$-i$	
3	y	z	d		a	b				b			c	$-d$		g				$-i$
4	y	x			b	a	d		$-d$	c			b			g		i		
5	z	x				d	a	b	c	$-d$		b					g	$-i$		
6	z	y		d			b	a	b		$-d$	c					g		i	
7	z	x				$-d$	c	b	a	d		b					g	i		
8	y	x			b	c	$-d$		d	a			b			g		$-i$		
9	x	y	b	c				$-d$			a	d		b	g				i	
10	z	y		$-d$			b	c	b		d	a					g		$-i$	
11	y	z	$-d$		c	b				b			a	d		g				i
12	x	z	c	b	$-d$						b		d	a	g					$-i$
13	x		e	e							e			e	h					
14	y				e	e				e			e			h				
15	z						e	e	e			e					h			
16		x				f	$-f$		f	$-f$								j		
17		y		$-f$				f			f	$-f$							j	
18		z	f		$-f$								f	$-f$						j

where $a = \dfrac{-Y}{2(4+Y)} + \dfrac{Z}{2(4+Z)}$; $\quad b = \dfrac{4}{2(4+Y)}$;

$c = \dfrac{-Y}{2(4+Y)} - \dfrac{Z}{2(4+Z)}$; $\quad d = \dfrac{4}{2(4+Z)}$;

$e = b$; $\quad f = Zd$;

$g = Yb$; $\quad h = \dfrac{Y-4}{Y+4}$;

$i = d$; $\quad j = \dfrac{4-Z}{4+Z}$ (2.12)

The subscripts of admittance Y and impedance Z are those of the corresponding stubs. For example, Y_x is the characteristic admittance of the ε_x stub normalized to Y_o, the characteristic admittance of free space. Z_x is the characteristic impedance of the μ_x stub normalized to Z_o, the characteristic impedance of free space. If we consider, for example, the element S_{48}, we note that port 4 and port 8 are associated with E_y and H_x, so S_{48} becomes

$$S_{48} = c = \frac{-Y_x}{2(4+Y_x)} - \frac{Z_y}{2(4+Z_y)}$$

The representation of space by the condensed node is shown in Figure 2-6 [1], where u, v, w are the unit cell dimensions in the x-, y-, z- directions, respectively.

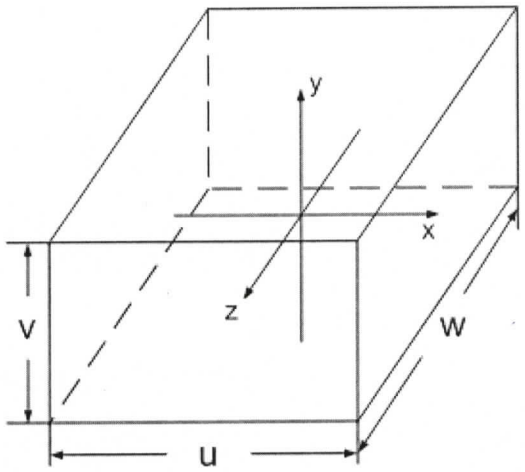


Figure 2-6 Representation of space by the condensed node.

The incident voltage impulses for exciting the six field components, Ex, Ey, Ez, Hx, Hy, Hz , can be calculated using Equation (2.13).

$$V_1^i = (uE_x + wZ_oH_z)/2; \qquad V_2^i = (uE_x - vZ_oH_y)/2;$$

$$V_3^i = (vE_y - wZ_oH_z)/2; \qquad V_4^i = (vE_y + uZ_oH_x)/2;$$

$$V_5^i = (wE_z - uZ_oH_x)/2; \qquad V_6^i = (wE_z + vZ_oH_y)/2;$$

$$V_7^i = (wE_z + uZ_oH_x)/2; \qquad V_8^i = (vE_y - uZ_oH_x)/2;$$

$$V_9^i = (uE_x + vZ_oH_y)/2; \qquad V_{10}^i = (wE_z - vZ_oH_y)/2;$$

$$V_{11}^i = (vE_y + wZ_oH_z)/2; \qquad V_{12}^i = (uE_x - wZ_oH_z)/2;$$

$$V_{13}^i = uE_x/2; \qquad V_{14}^i = vE_y/2;$$

$$V_{15}^i = wE_z/2; \qquad V_{16}^i = -Z_xZ_0uH_x/2;$$

$$V_{17}^i = -Z_yZ_0vH_y/2; \qquad V_{18}^i = -Z_zZ_0wH_z/2 \qquad (2.13)$$

On the other hand, the field components resulting from various voltage impulses can be calculated with Equation (2.14). In the 1990's, more elegant symmetrical forms of the scattering matrix were proposed ([12-15]).

$$
\begin{aligned}
E_x &= 2(V_1^i + V_2^i + V_9^i + V_{12}^i + Y_x V_{13}^i)/u(4+Y_x) \\
E_y &= 2(V_3^i + V_4^i + V_8^i + V_{11}^i + Y_y V_{14}^i)/v(4+Y_y) \\
E_z &= 2(V_5^i + V_6^i + V_7^i + V_{10}^i + Y_z V_{15}^i)/w(4+Y_z) \\
H_x &= 2(V_4^i - V_5^i + V_7^i - V_8^i - V_{16}^i)/Z_o u(4+Z_x) \\
H_y &= 2(-V_2^i + V_6^i + V_9^i - V_{10}^i - V_{17}^i)/Z_o v(4+Z_y) \\
H_z &= 2(-V_3^i + V_1^i + V_{11}^i - V_{12}^i - V_{18}^i)/Z_o w(4+Z_z)
\end{aligned}
\tag{2.14}
$$

2.2.2 Generalized Symmetrical Condensed Node TLM

In 1996, the 3D Generalized Symmetrical Condensed Node (GSCN) was developed to model more general media [16] . The 3D GSCN features six different link lines with different characteristic impedances, three open circuit stubs, three short circuit stubs and six lossy elements for modeling electric and magnetic losses which are not included in the SCN derivation presented in [17].

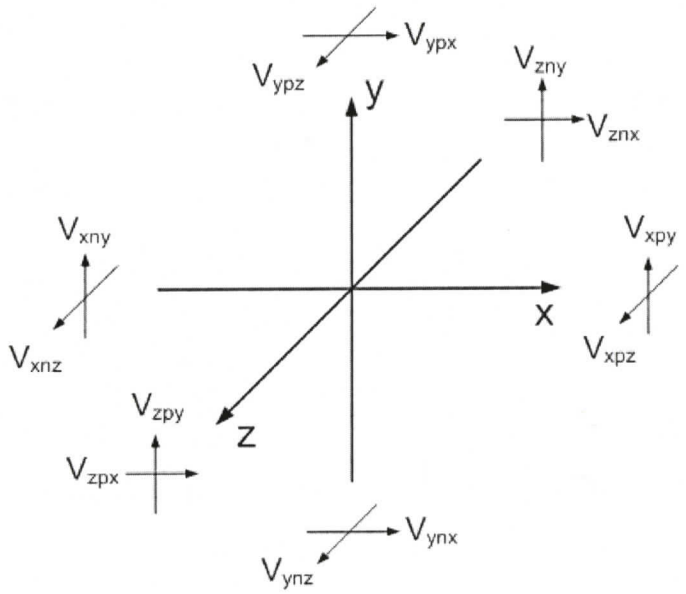


Figure 2-7 The impulse labelling scheme of the 3D GSCN node.

Compared to 3D SCN, 3D GSCN adds a new capability that enables the node to handle lossy material. As shown in Figure 2-7, 3D GSCN uses a new labelling scheme to represent voltage impulses instead of using $V_1 - V_{12}$ to represent voltage impulses as in Figure 2-5 [18] . The GSCN in Figure 2-7 defines voltage impulses by direction and polarization of link lines. The first letter in the subscript is the direction of the link line; the second letter n or p in the subscript indicates that the voltage is incident from the negative or positive side of the node, respectively; and the third letter is the polarization of the link line. For example, V_{zpy} means a voltage pulse is coming from the positive side of the node, along a z-directed, y-polarized link line. V_{oi} and V_{si} are voltages on open circuit stubs and short circuit stubs, respectively. V_{ei} and V_{mi} are voltages on electric and magnetic loss elements, respectively.

The reflected voltages in the GSCN scattering procedure (see [19, 20]), are given in Equation (2.15) [16] . The voltages reflected into the open circuit stubs, short circuit stubs, and lossy elements are shown in Equation (2.16). Lossy elements are described in Equation (2.17). The equivalent voltages and currents that model the electric and magnetic field are given in Equation (2.18) [21] :

$$V_{inj}^r = V_j \pm I_k Z_{ij} - V_{ipj}^i$$

$$V_{ipj}^r = V_j \pm I_k Z_{ij} - V_{inj}^i \tag{2.15}$$

$$V_{oi}^r = V_i - V_{oi}^i; \qquad V_{si}^r = I_i Z_{si} + V_{si}^i; \qquad V_{ei}^r = V_i; \qquad V_{mi}^r = R_{mi} I_i \tag{2.16}$$

$$G_{ei} = \sigma_{ei} \frac{\Delta j \Delta k}{\Delta i}; \qquad R_{mi} = \sigma_{mi} \frac{\Delta j \Delta k}{\Delta i} \tag{2.17}$$

where σ_{ei} and σ_{mi} are equivalent electric and magnetic conductivities in the i-direction.

$$V_i = \frac{2Y_{ki}(V_{kni}^i + V_{kpi}^i) + 2Y_{ji}(V_{jni}^i + V_{jpi}^i) + 2Y_{oi}V_{oi}^i}{2(Y_{ki} + Y_{ji}) + Y_{oi} + G_{ei}}$$

$$I_i = \frac{2(V_{jpk}^i - V_{jnk}^i + V_{knj}^i - V_{kpj}^i - V_{si}^i)}{2(Z_{jk} + Z_{kj}) + Z_{si} + R_{mi}} \tag{2.18}$$

where $(i,j,k) \in \{(x,y,z),(y,z,x),(z,x,y),(x,z,y),(y,x,z),(z,y,x)\}$.

2.3 Common Features in the Various TLM Methods

The theories of the 2D Shunt Node, 2D Series Node, 3D SCN and 3D GSCN have been summarized in this chapter. Although these nodes have different scattering matrices, the modeling procedure is similar. Once a voltage impulse reaches a node at time $t = k\Delta t$, scattering occurs. The reflected voltage impulses travel along the link lines and arrive at adjacent nodes at $t = (k+1)\Delta t$.

Like other numerical methods, TLM is subject to various sources of error such as truncation error, velocity or dispersion error, and coarseness error. These errors depend on the total calculation time, the direction of wave propagation, with respect to the mesh axes, the frequency and the mesh size. Error correction techniques, as described in [2] can be used to improve the accuracy, and hence efficiency, of the TLM method.

2.4 Numerical Convolution for Two-Dimensional TLM Diakoptics

Diakoptics is a technique that involves breaking down a large structure into substructures that can be solved independently before reconnecting them together [22]. The technique was first proposed by Kron [4] in 1963, and it was subsequently applied in frequency domain [23-25] and time-domain methods for TLM modeling in 1981 [26, 27]. Eswarappa and Righi applied this technique to create wide-band dispersive absorbing boundary conditions [28, 29].

2.4.1 Response of a TLM Structure to a Single Impulse Excitation

Figure 2-8 shows a top view of a section of a rectangular waveguide operating at TE_{10} mode modeled by a 2D TLM mesh. The input branches (removed branches) in the reference plane are labelled from *n= 1 to N*, while the output branches are labelled from *m=1 to M* or *p=1 to P* ($M = N = P$) [28]. The impulse response of a TLM structure is called Johns Matrix [2]. A 3D reflection Johns Matrix term $g_{11}(m,n,k)$ represents the response of a TLM structure at output branch m, over time $t = k\Delta t$ to a single impulse excitation at input branch

n, occurring at time-step $t = 0$. A 3D transmission Johns Matrix term $g_{21}(p,n,k)$ represents the response of a TLM structure at output branch p, over time $t = k\Delta t$ to a single impulse excitation at input branch n, occurring at time-step $t = 0$ [22].

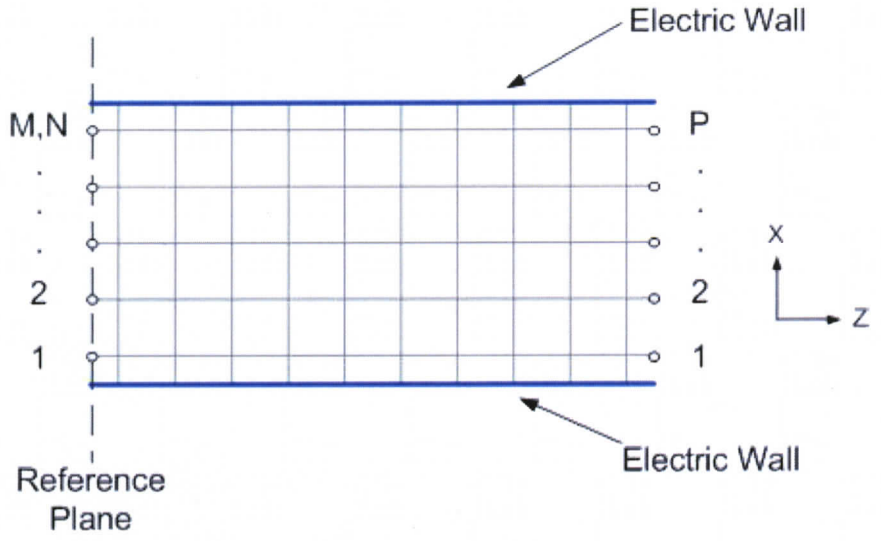


Figure 2-8 Top view of a section of a rectangular waveguide operating at TE_{10} mode modeled by a 2D TLM mesh.

2.4.2 Response of a TLM Structure to Arbitrary Impulse Excitations

The reflected voltage impulses $V^r(m,k)$ can be found by the numerical convolution of incident voltage impulses $V^i(n,k')$ with the Johns Matrix as shown in Equation (2.19) for k'=0:

$$V^r(m,k) = G(m,n,k-k') * V^i(n,k') \tag{2.19}$$

The convolution process was described by Hoefer in Equation (2.20) [22] .

$$V^r(m,k) = \sum_{n=1}^{N}\sum_{k'=0}^{k} g(m,n,k-k')\cdot V^i(n,k') = \sum_{n=1}^{N}\sum_{k'=0}^{k} g(m,n,k')\cdot V^i(n,k-k') \tag{2.20}$$

The convolution process is a simple multiply-shift-add procedure which is shown in Figure 2-9 [2]. $V^i(n,k)$ is an arbitrary impulse excitation of a TLM structure as shown in Figure 2-9 (a). The impulse response of the TLM structure is $G(m,n,k)$ as shown in Figure 2-9 (b). The response of the first impulse in $V^i(n,k)$ labelled "1" in Figure 2.9 (a) is the first figure in Figure 2.9 (c) labelled

by "1", which is calculated as the amplitude of the first impulse times the response of the whole structure $G(m,n,k)$. Similarly, the response of the second impulse in $V^1(n,k)$ is the second figure in Figure 2.9 (c), which is calculated as the amplitude of the second impulse times $G(m,n,k)$ but with one-time-step delay.

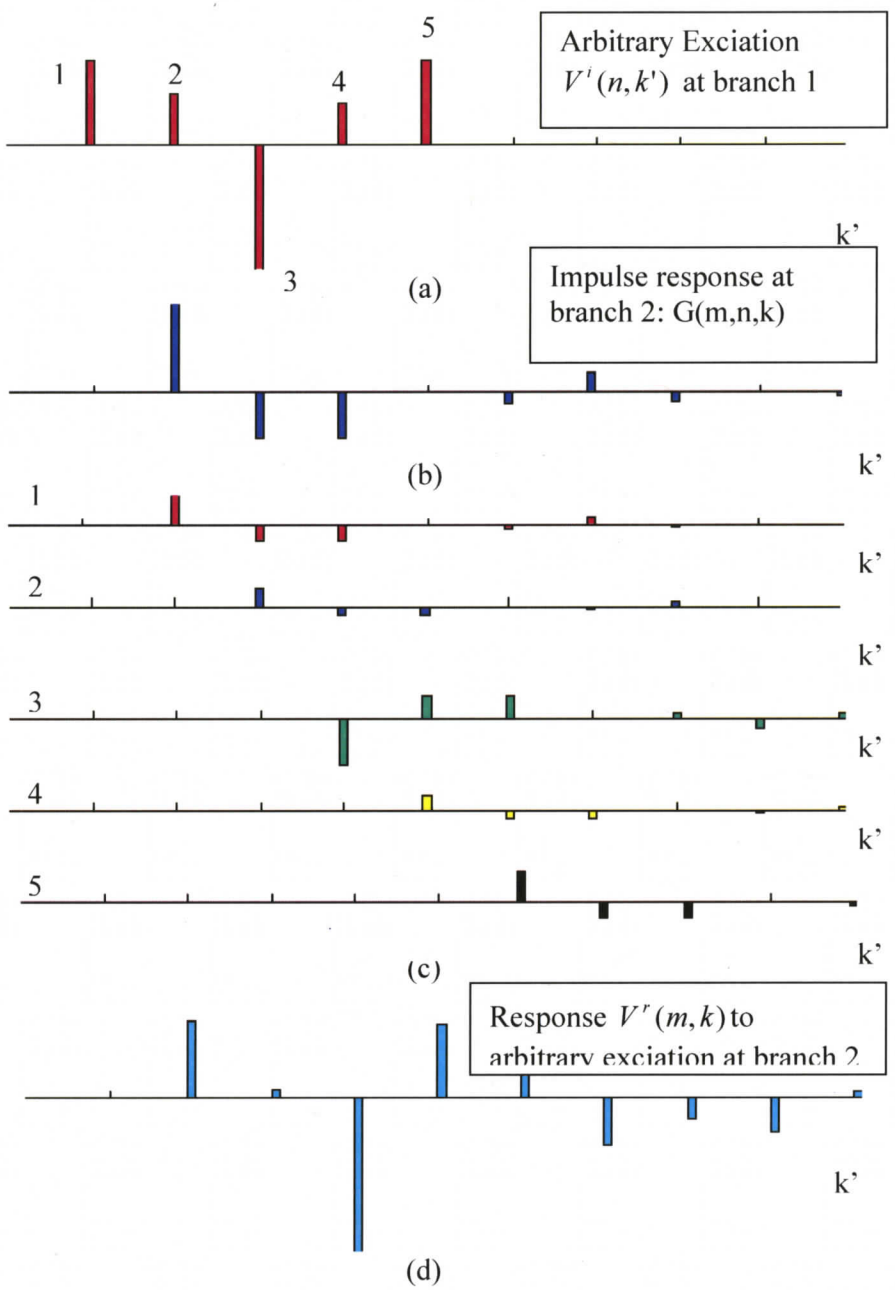


Figure 2-9 The numerical convolution procedure to obtain the response to arbitrary excitation.

With the same method, the responses of all the impulses in $V^i(n,k)$ can be found in Figure 2.9 (c). The response of a TLM structure is the sum of these individual responses. The details of the convolution procedure can be found in [2] and [30]. After the Johns matrix of a TLM structure is obtained, the response of a TLM structure $V^r(m,k)$ to arbitrary excitations $V^i(n,k')$ is found by Equation (2.19) and (2.20).

2.4.3 Analogy between Scattering Matrix and Johns Matrix

The Johns matrix in the time domain is analogous to the scattering matrix in the frequency domain. In the case of a structure with a single interface, the Johns Matrix has the characteristics of a reflection coefficient matrix, and reflected impulses are computed by convolving the incident pulses with the impulse response $[G]$. This corresponds to the computation of the reflected wave in the frequency domain by complex multiplication of the incident wave with the reflection coefficient S_{11}. The analogy between scattering matrix and Johns Matrix is shown Table 2-1.

	Frequency Domain	Time Domain
1 – port	$b_1 = S_{11} \bullet a_1$	$v_1^r = G_{11} * v_1^i$
2 – port	$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \bullet \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$ Complex multiplication	$\begin{bmatrix} v_1^r \\ v_2^r \end{bmatrix} = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} * \begin{bmatrix} v_1^i \\ v_2^i \end{bmatrix}$ Convolution

Table 2-1 Analogy between Scattering Matrix and Johns Matrix.

Chapter 3

Design and Implementation of the Co-Simulation framework

This chapter gives an overview of the design and implementation of a co-simulation framework for interconnecting PSpice circuits with a time-domain electromagnetic field simulator. An introduction to the co-simulation framework design will be presented first, along with the motivation behind this work. The detailed class designs for the co-simulation framework will be presented in this chapter. Implementation details are attached in Appendix A and B.

3.1 Introduction

Commercial electromagnetic (EM) software packages are used to handle high frequency simulations of radio frequency (RF) circuits and are often based on either frequency- or time-domain methods. Due to the increasing frequency and complexity of RF circuits, there has been heightened interest for a co-simulation framework to connect EM simulators with circuit simulators.

Electromagnetic field solvers such as CST Microwave Studio and Flomerics Microstripes have proprietary links to Agilent ADS and AWR Microwave Office, respectively; however, popular electronics circuit solvers such as Cadence PSpice cannot handle the co-simulation of field and circuit. A co-simulation framework is developed to interface PSpice with an electromagnetic field simulator based on the Transmission Line Matrix (TLM) method.

One basic approach to couple the TLM structure with circuit simulation is to directly couple field and circuit equations using a global simulation [31, 32]. This approach reduces flexibility with respect to the development of individual simulators. Another approach is to couple EM fields and a circuit using the

Scattering Parameters (S-parameters) of the circuit. Complex S-parameters can be introduced into PSpice [33, 34] in the form of an Analog Behavioural Module (ABM). The time-domain response of this module is then found from the S-parameters by inverse Fast Fourier Transform, followed by its convolution with the outside simulus. However, transforming a signal between the frequency domain and time domain using a Fourier transform then an inverse Fourier transform may induce significant errors. The resulting response may be non-causal due to an insufficient number of frequency domain data. Instead of using S-parameters of the circuit, fields and circuits can be coupled in the time domain by an exchange of data at each time-step. This approach is more direct, robust and flexible than the S-Parameter representation combined with the Inverse Fourier Transform.

In this chapter, the design and implementation of a co-simulation framework for interconnecting TLM meshes with PSpice circuits are presented. The TLM mesh is converted to a Thévenin equivalent source V_{TLM} and a Thévenin equivalent impedance Z_{TLM} in order to enable coupling with PSpice circuits. V_{TLM} is used to represent the voltage emerging from the TLM mesh and is updated at each time-step. Z_{TLM} depends on the TLM link line impedance at the TLM-PSpice interface. At each time step, the PSpice engine is invoked to do one time-step calculation in the circuit. PSpice returns the output results back to the TLM mesh for another time-step calculation. This process is repeated until the end of the computation. If the impulse response of the TLM structure can be pre-calculated, the Thévenin equivalent source V_{TLM} is calculated by the convolution of the impulse response of the TLM structure and reflected voltage returned from the PSpice circuit. This convolution approach further improves simulation speed of the TLM-PSpice co-simulation method.

3.2 Design Methodology

The proliferation of wireless communication has driven up the demands for electromagnetic and circuit co-simulation platforms. An example which would benefit from the use of field and circuit co-simulation is for impedance matching and tuning, such as with single stub and double stub tuning. A tuning network is needed to match a load circuit to the transmission line in order to ensure that power flow is maximized. The load circuit model can be realized in a circuit simulator, such as PSpice; the transmission line can be simulated in an electromagnetic (EM) field solver, such as MEFiSTo, which is based on the TLM method. An interconnection framework is needed to exchange data between the field simulator and PSpice at each time step as shown in Figure 3.1. Another common problem is to co-simulate an antenna and its feeding circuit. An antenna can be simulated by the field simulator MEFiSTo. Data are exchanged between the field simulator and PSpice using the interconnection framework. To make the interconnection framework easy to modify and reuse, the framework is developed with Object-Oriented Programming (OOP).

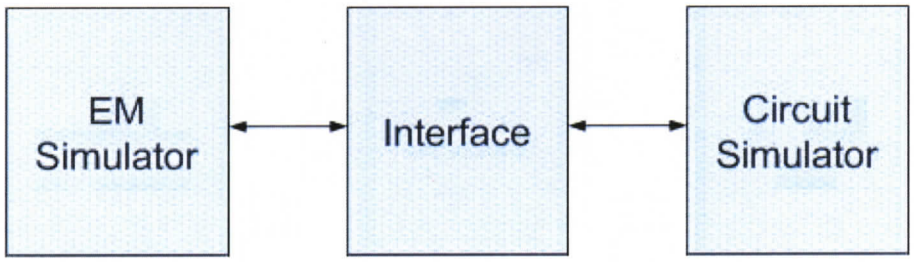


Figure 3-1 Interconnection of EM simulator and circuit simulator.

Two key characteristics of OOP are encapsulation and inheritance. With encapsulation, an outside program can use an object through inputs and outputs to the object, without the need to understand the internal implementations of the object. These internal object implementations are not automatically accessed by the core program.

Another important characteristic of OOP is inheritance. Inheritance is a process by which a related class called "derived class" is created from an existing base class. The derived class inherits some functions and variables from the base class, except for constructors, destructors and friends of the base class. In addition, the derived class can also add its own functions and variables. It is possible for a class to inherit data and functions from more than one base class. This feature is called multiple inheritance. Polymorphism is an ability to call different functions by using the same function handle, which takes OOP to its full potential. The benefit of using OOP for the program is that it allows for ease of maintenance via classes, the use of which greatly simplifies the upgradability of the program.

Relationships between classes for the interconnection framework are shown in Figure 3.2. The EM-simulator is based on the TLM method. It handles simulations when fundamental Transverse Electromagnetic (TEM) and Transverse Electric (TE) modes are present in the structure, which are implemented using class *TLM* and class *TLM_TE*, respectively. Both of these classes include functions for excitation, scattering, connection, and boundary procedures and have been described previously in Chapter2. Class *TLM_TE* is derived from class *TLM* with redefined functions for excitation and boundary procedures since the field profile at the boundary of the structure are different for TEM mode and TE_{10} mode.

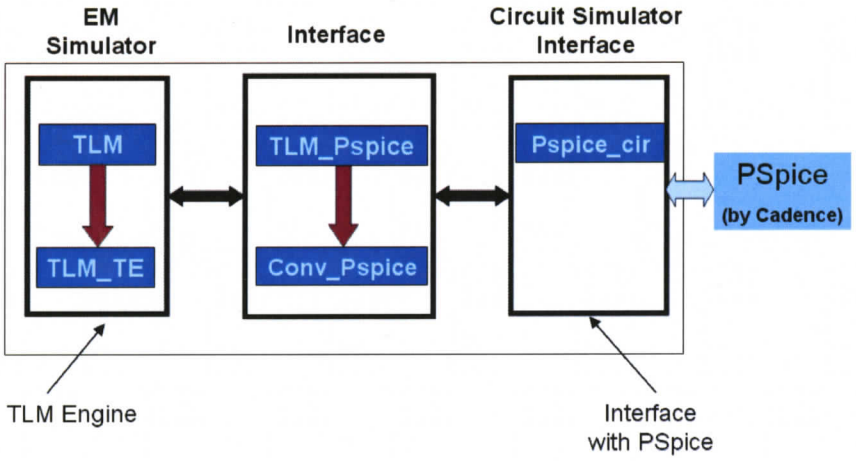


Figure 3-2 Interconnection class relationships for the co-simulation framework.

PSpice is a commonly used circuit simulation software package, and is one of the commercial derivatives of Simulation Program with SPICE. Class *Pspice_cir* handles all the simulations implemented by PSpice. The heart of the PSpice is the netlist file. A netlist file (.cir) contains a list of electrical components and the nets, or nodes for connecting components together. A netlist file also indicates initial conditions for some electrical components, such as capacitors and inductors. At the end of a netlist file, analysis requirements, such as those defined in transient analysis or AC analysis, are defined. Once a PSpice netlist file is created, the PSpice engine can be invoked by running the command line:

psp_cmd inputXXX.cir

The output analysis results are then saved as inputXXX.out within the same local folder as the circuit file.

Classes *TLM_Pspice* and *Conv_Pspice* exchange data between the EM simulator and the circuit simulator at each time-step. Both of these classes include functions for updating the Thévenin equivalent source V_{TLM} of the TLM structure and functions for calculating the voltages returned to the TLM mesh. Class *Conv_Pspice* is derived from class *TLM_Pspice*. If the TLM structure is characterized by a Johns Matrix, the PSpice circuit can be connected to the TLM structure by convolution. Class *Conv_Pspice* also handles the convolution process.

3.3 Design and Implementation

3.3.1 Class *Array3D* Implementation

When a set of classes have the same functionalities and implementations but contain different data types, it is a waste of code to repeat the entire class for each data type. Class Template allows functions and classes to operate with different data types.

Class *Array3D* is a generic Three-Dimensional (3D) array class template shown in Figure 3-3 by object browser in Visual Studio. (Object browser icons in this chapter are shown in Table 3-1.) Objects of this class can be treated as three-,

two- and one-dimensional arrays. Array elements can be accessed using the ( ) operator. Data types for array elements can be of type double, integer or character. The *Array3D* class includes functions for array memory allocation, deallocation, array copy, initialization and so on.

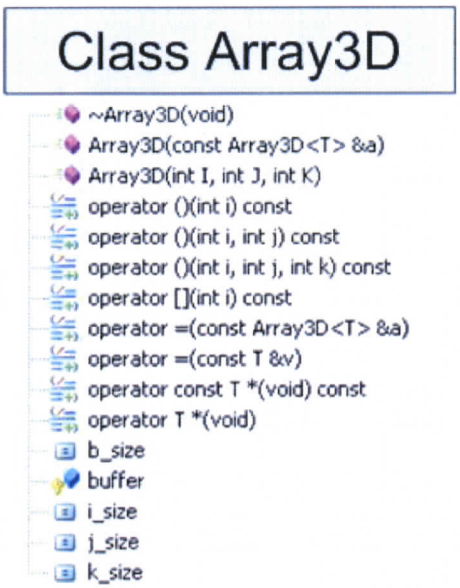


Figure 3-3 Class *Array3D* implementation.

Examples 1 through 3 below show how to use the class *Array3D*. Example 1 is a 1D array example. Array "a" is a 1D array with size 5. All the elements in the array have data type character (char). This is not possible in the traditional C/C++ syntax. In the second line, all the elements in array "a" are assigned with character 'c'. The third element of the array is changed to character 'm' in the last line.

```
Example 1: 1D array
Array3D<char> a(5,1,1);
a = 'c';
a(2) = 'm';
```

	Global functions
	Operator
	Constant
	Public variables
	Protected variables
	Private variables

Table 3-1 Object browser icons in Visual Studio

Example 2 gives a 2D array example. Array "b" and "c" are two 2D arrays of type double with the same size 10x10. All the elements in array "b" are initialized to zero, and are copied to array "c". In the last line, one element in array "b" is changed to 8.

```
Example 2: 2D array
Array3D<double> b(10,10,1), c(b);
b=0;
c=b;
b(2,5)=8;
```

Example 3 gives a 3D array example. Array "d" is a 3D array of integers (int) with size 10x10x10 and is initialized to zero. One element in array "d" is changed to 3 in the last line.

```
Example 3: 3D array
Array3D<int> d(10,10,10)
d=0;
d(1,1,1)=3;
```

3.3.2 Class *TLM* and *TLM_TE* Implementation

The two-dimensional (2D) TLM Method is used to model three-dimensional (3D) wave problems where all three field components depend on two spatial dimensions and have no variation in the third spatial dimension [2]. A *TLM* class handles electromagnetic field simulations in the fundamental TEM mode. A *TLM_TE* class is derived from the *TLM* class and it handles electromagnetic field simulations in the TE_{10} mode. This section gives an overview of the implementation of 2D classes *TLM* and *TLM_TE* whose

relationships are shown in Figure 3-4. A *TLM* class always contains member functions for excitation, scattering, connection, boundary conditions, calculating Johns responses G_{11} and G_{12}, and calculating node voltages at a given node location.

In the building block of a 2D TLM shunt node shown in Figure 2-3, a node is associated with four link lines, which are labelled from 1 to 4. Incident and reflected voltage impulses at four link lines are stored in the 3D array. $Vi1(z,x,t)$ to $Vi4(z,x,t)$ are voltage impulses incident on each node. $Vr1(z,x,t)$ to $Vr4(z,x,t)$ are scattered voltage impulses reflected back from each node, where z and x represent the location of the node in the mesh, and t represents the time.

The *TLM* class contains a constructor, which is used to initialize variables. This constructor has three inputs: the number of nodes in the z- and x-direction, the *zSize* and *xSize*, respectively, and the total number of simulation time steps *totalTime*. Within this constructor, the sizes of incident voltage impulses $Vi1(z,x,t)$ to $Vi4(z,x,t)$ and reflected voltage impulses $Vr1(z,x,t)$ to $Vr4(z,x,t)$ are defined, and all voltage impulses are initialized to zero. In order to compare results from the proposed algorithms with results from the field simulator MEFiSTo-2D, node voltages at a specific node location are saved to a text file *node.txt*. The text file is cleared once an object of this class is created in order to avoid clearing the text file manually at the end of each simulation.

In the "excitation" function, a unit impulse is injected into the TLM mesh at node location (0, x), where x is from 0 to the total number of nodes in the x-direction minus one, or *xSize - 1*. "Scattering" and "Connection" are two basic procedures of the TLM algorithm. "Scattering" occurs when a unit voltage impulse reaches the node center. The input of the scattering function is the number of simulation time steps, k. The implementation of the scattering function is based on Equation (2.3) of Chapter 2. During the "Connection" procedure, incident impulses are exchanged with voltage impulses from neighbouring nodes [35] . The "connection" function in the TLM class is implemented based on Equation (2.4) of Chapter 2. If a node is located at the boundary of the mesh,

boundary conditions should also be applied to that node. The inputs of the "boundary" function are the reflection coefficients of four structure boundaries. If a TLM structure is connected to another TLM structure or a PSpice circuit, voltage impulses from another TLM or PSpice circuit become inputs of the "boundary" function.

Transmitted voltage impulses at the right-most boundary of the TLM structure are calculated by the function *transG12( )*. Reflected voltage impulses at the left-most boundary of TLM structure are calculated by the function *reflectG11( )*. If unit impulses are incident to a TLM mesh, the Johns responses G_{11} and G_{12} of the TLM structure can also be calculated using these two functions. To compare results from the proposed algorithm with results from the field simulator MEFiSTo-2D, the node voltages at a specific location in the TLM mesh are calculated using both methods. The node voltage of the proposed algorithm is calculated using the function *cal_vnode( )*, which is based on Equation (2.2) in Chapter 2.

The class *TLM* handles electromagnetic field simulations in the TEM mode. The class *TLM_TE* is the class *TLM*'s derived class, which handles electromagnetic field simulations in the TE_{10} mode. To excite the TE_{10} mode, voltage impulses with a half-sine distribution are injected into all branches of a one-port of the TLM mesh at the beginning of the simulation, instead of unit impulses. Voltage impulses traveling in the x-direction (refer to the x, z coordinate system of Figure 2.3) have a half-sine distribution in the waveguide cross-section. "Scattering" and "Connection" procedures in the TE_{10} mode are the same as those in the TEM mode, but "Excitation" and "Boundary" procedures are different. Class *TLM_TE* redefines functions for "Excitation" and "Boundary" procedures, and inherits functions for "Scattering" and "Connection" procedures from the *TLM* class. Class *TLM_TE* also includes the function *cal_vnode( )* for calculating node voltages at another location in the TLM mesh; and *cal_sum( )* for calculating the sum of voltage impulses with a half-sine distribution at the interface cross-section, as based on Equation (4.11) of Chapter 4.

Figure 3-4 Classes *TLM* and *TLM_TE* implementations.

3.3.3 Class PSpice_cir Implementation

PSpice is circuit simulation software derived from SPICE. A netlist file gives a description of a circuit which can be used to invoke PSpice's engine to allow PSpice to start the circuit analysis. A netlist file contains a list of electrical components and the nets, or nodes, for interconnecting those components. Class *PSpice_cir* handles implementations involved with PSpice (shown in Figure3-5), which include functions for storing the PSpice netlist file to a 2D array, updating PSpice netlist files, invoking the PSpice engine, and calculating total voltages at the PSpice and TLM interface for each time-step.

Class *PSpice_cir* includes three overloading constructors. Inputs to each constructor indicate the names of components whose initial conditions need to be changed at each time step, and also the name of the netlist file for invoking the PSpice engine. In each constructor, the position of each variable to be changed is recorded in order to be used in other functions within the class. Once the object of Class *PSpice_cir* is created, the netlist file is read and saved to a 2D array. To update the netlist file, only the corresponding elements in the 2D array need to be updated.

The Thévenin equivalent voltage source V_{TLM} and the internal resistance Z_{TLM} are defined for each TLM mesh. V_{TLM} is updated at each time-step, since voltage impulses emerging from the TLM mesh are updated at each time-step. If a PSpice circuit is connected to one TLM structure, then V_{TLM} in the PSpice circuit is updated using the function *update(string)*, where the input to the function is the updated V_{TLM}. If the PSpice circuit is connected to two TLM structures, the V_{TLM} for both TLM structures are updated using the function *update2(string, string)*, where the inputs to the function are the updated V_{TLM} for both TLM structures. Capacitors and inductors are energy storing components, which need the results from the previous time-step. If capacitors or inductors are present in the circuit, initial conditions need to be changed at each time-step.

After the PSpice netlist file is updated, the new circuit file is ready to invoke the PSpice engine. The function *calculate( )* includes the commands for

invoking the PSpice engine and for calculating the total voltage at the TLM and PSpice interface. The function *calculate( )* returns the total voltage result to the TLM.

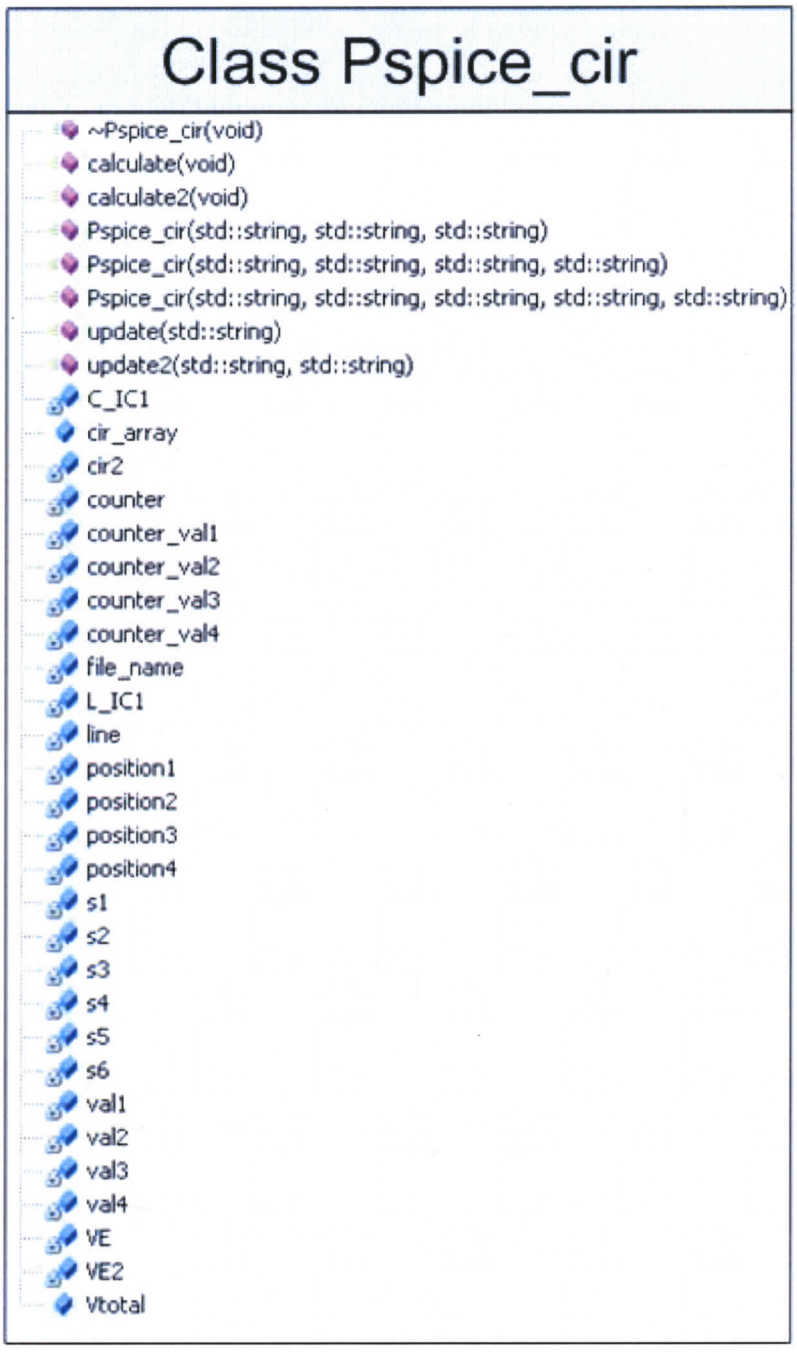


Figure 3-5 Class *Pspice_cir* implementation.

3.3.4 Class TLM_Pspice and Conv_Pspice implementation

Implementations of Classes TLM_Pspice and Conv_Pspice are shown in Figure 3-6. Class *TLM_Pspice* handles implementations at the interface of TLM structures and PSpice circuits. It contains two functions: *toPspice(double, double)* and *toTLM(double)*. The function *toPspice(double, double)* converts voltage impulses from the TLM mesh to the Thévenin equivalent voltage source in the PSpice circuit. The function *toTLM(double)* converts the total voltage calculated by PSpice to the reflected voltages returning to the TLM mesh.

Class *Conv_Pspice* is derived from class *TLM_Pspice*, so it inherits all accessible members from the base class. Class *Conv_Pspice* adds two other functions for performing convolutions. The function *Convolve_array* handles convolutions of two arrays and is based on Equation (2.20). The inputs of the function are two arrays and the size of the convolution output of the two arrays. The output of the function is the convolution results of the two arrays. The inputs for the function *Convolve* are a number, an array, size of the output and the time-step, whereas its output is the convolution results.

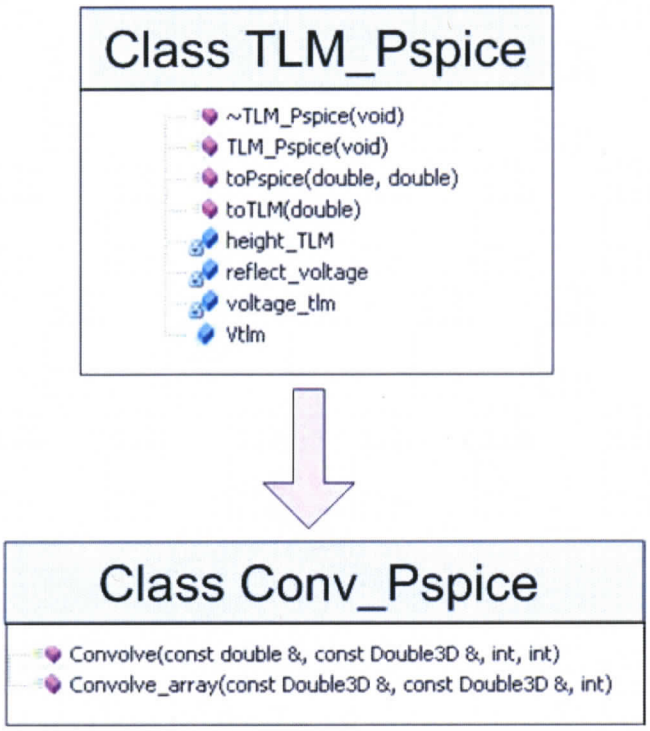


Figure 3-6 Classes *TLM_Pspice* and *Conv_Pspice* implementations.

3.4 Class Utilization

3.4.1 Class *TLM* and Class *TLM_TE* Utilizations

As indicated in Figure 3-2, there exist two classes in the EM simulator, class *TLM* and class *TLM_TE*, which handle electromagnetic field simulations in the fundamental Transverse Electromagnetic (TEM) mode and Transverse Electric (TE_{10}) mode, respectively. The field simulations are modeled by the two-dimensional (2D) Transmission Line Matrix (TLM) method. A *TLM* class contains member functions for excitation, scattering, connection, boundary conditions, calculating Johns responses G_{11} and G_{12}, and calculating node voltages at a given node location.

Class *TLM_TE* is derived from class *TLM*. Unit impulses are injected into all the branches at the input port of a TLM mesh in order to excite the TEM mode; alternately, a field profile with a half-sine distribution can be injected into the input port of a TLM mesh in order to excite the TE_{10} mode. In short, class *TLM_TE* contains the redefined "excitation" function. For a waveguide operating

in the TEM mode, the field profile at the waveguide's cross-section is uniform; for a waveguide operating in the TE_{10} mode, the field profile at the waveguide's cross-section is a half-sine distribution. Therefore, class *TLM_TE* contains a redefined *boundary()* function. In addition, class *TLM_TE* also contains the function *cal_sum()*, which is used for extracting the fundamental TE_{10} mode when higher order modes are present in the structure. Both class *TLM* and class *TLM_TE* contain constructors which define the size of the TLM structures (*zmax*, *xmax*) and the total simulation time-step (*tmax*).

Below are two examples that describe how to use class *TLM* and class *TLM_TE*. Example1 shows how to acquire Johns response G_{11} and G_{12} of a waveguide with size $20 \times 20\Delta\ell$ operating in the TEM mode. The results are saved as two one-dimensional arrays, g11 and g12. Example2 demonstrates how to realize node voltages at the node location (2,2) in a 2D TLM mesh for a waveguide operating in the TE_{10} mode. The node voltages are saved to the text file "node.txt".

```
//Example1: TEM mode
zmax = 20; xmax = 20; tmax = 1000;
TLM TLM1(zmax, xmax, tmax); //Define 2D TLM mesh size and total simulation time
TLM1.exitation(1.0); //Excitation: inject unit voltage impulses to input port of TLM
                     //mesh
for (int k = 0; k < tmax; k++)
{
     TLM1.scattering(k); //Scattering process
     g12[k+1] = TLM1.transG12(); //Calculate transmited Johns response G12
     g11[k+1] = TLM1.reflectG11(); //Calculate reflected Johns response G11
     TLM1.connection(); //Connection process
     TLM1.boundary(1,0,1,-0.17157); //Reflection coefficients of boundary
}


//Example2: TE10 mode
zmax = 20; xmax = 20; tmax = 1000;
TLM_TE TLM1(zmax, xmax, tmax); //Define 2D TLM mesh size and total simulation time
TLM1.exitation(11); //Excitation: inject a field profile with a half-sine
                    //distribution to the input port of the TLM mesh in order to
                    //excite the TE10 mode.
                    //Input: number of removed branches at the input port (11)
for (int k = 0; k < tmax; k++)
{
     TLM1.scattering(k); //Scattering process
     TLM1.cal_vnode(2,2); //Save node voltages at location (2,2) in the 2D TLM mesh
                          //to the text file "node.txt"
     TLM1.connection(); //Connection process
```

```
    TLM1.boundary(-1,0,-1,-0.17157); //Reflection coefficients of boundary
}
```

3.4.2 Class *Pspice_cir* Utilization

Class Pspice_cir handles all the calculations involved with the circuit simulator PSpice. This class includes functions for storing PSpice netlist files to 2D arrays using three constructors, updating netlist files using the functions *update(string)* and *update2(string, string)*, invoking the PSpice engine and calculating the total voltage at the interface between PSpice and the TLM engine at each time step by employing functions *calculate(void)* and *calculate2(void)*.

If there is a single voltage source in a netlist file, the file is updated using the function *update(string)*; if there are two voltage sources in a netlist file, the file is updated using the function *updated(string, string)*. Once the netlist file is updated, the new circuit file is ready to invoke the PSpice engine. Functions *calculate(void)* and *calculate2(void)* include commands for invoking the PSpice engine and calculating the total voltage at the interface of the TLM engine and PSpice. The function *calculate(void)* is used when inductors are present in the circuit file and function *calculate2(void)* is used when inductors are not present in the circuit file.

A sample netlist file which includes the Thévenin equivalent source of a TLM mesh (V_s and R_s) and a PSpice circuit (R_1, C_1 and L_1) is shown below. Since inductors and capacitors are energy storing components, their initial conditions must be updated at each time step. The TLM engine exchanges data with the PSpice circuit at each time step, so the value of V_s must also be changed at each time step. In the netlist file, the value of V_s, the initial conditions of capacitor C_1 and the initial conditions of inductor L_1 are updated at each time step.

```
** Sample netlist file: saved as input.cir
Vs    1     0     DC      0
Rs    1     2     26.6579
R1    2     0     5
C1    2     0     20p     IC=0
L1    2     3     20p     IC=0
V     3     0     0
```

```
* ANALYSIS
.TRAN          2.35865ps  2.35865ps 0us 2.35865ps UIC

* VIEW RESULTS
.PRINT         TRAN    V(1) V(2)  I(V)
.probe
.END
```

Below is an example that describes how to update the above netlist file using functions in class *Pspice_cir*:

```
//An example that describes how to use class Pspice_cir:
Pspice_cir circuit1("Vs", "C1", "L1","input"); //In the netlist file, three variables
                                               //need to be changed; the name of the
                                               //netlist file is input.cir.
circuit1.update(Vs.c_str()); //There is one voltage source, so use function
                             //update(string) to update the netlist file.
sum = circuit1.calculate(); //There is an inductor, so use function calculate() to
                            //calculate the total voltage at the interface of TLM
                            //mesh and Pspice
```

3.4.3 Class *TLM_Pspice* and Class *Conv_Pspice* Utilizations

Class *TLM_Pspice* and class *Conv_Pspice* are used to exchange data at the interface of TLM structures and PSpice circuits. Class *TLM_Pspice* handles direct connections between TLM structures and PSpice circuits by exchanging voltage impulse data computed from the TLM structure at each time step, whereas class *Conv_Pspice* inherits from class *TLM_Pspice* and handles connections between TLM structures and PSpice circuits by convoluting the pre-calculated Johns response of the TLM structure and the voltages reflected from PSpice to the structure.

Class *TLM_Pspice* contains two functions, *toPspice(double, double)* and *toTLM(double)*. The function *toPspice(double, double)* is used to calculate the Thévenin equivalent sources of the TLM structures for the PSpice netlist files at each time step. The two inputs of the function are the height of the TLM structure and the voltage impulses emerging from the TLM structure. The function *toTLM(double)* is used to calculate voltages going back to the TLM structure based on the function input, the total voltage at the interface of the TLM structure and the PSpice circuit. The example below describes how to use class *TLM_Pspice*:

```
//An example that describes how to use class TLM_Pspice:
TLM_Pspice interface1; //Interface1 is the object of class TLM_Pspice
Vs = interface1.toPspice(trans[k+1],1); //Calculate thevenin equivalent source of TLM
                                        //structure for PSpice circuit; trans[k+1]
                                        //are voltage impulses emerging from TLM
                                        //structures; the height of the TLM
                                        //structure is 1dl.
Vr = interface1.toTLM(sum); //Calculate voltages ging back to the TLM structure from
                            //PSpice; the input of the function is the total voltage
                            //at the interface of TLM structure and PSpice.
```

Class Conv_Pspice inherits from class TLM_Pspice, meaning that it inherits all the accessible members from its base class. Class Conv_Pspice adds two other functions, Convolve() and Convolve_array() for performing convolutions. The example below describes how to use the functions in class *Conv_Pspice*:

```
//An example that describes how to use class Conv_Pspice:
Conv_Pspice conv; //conv is the object of class Conv_Pspice
result = conv.Convolve_array(myout,g12,t1); //myout and g12 are two 1D arrays need to
                                            //be convolved; result is convolution
                                            //result; t1 is the size of the
                                            //convolution result.
```

Chapter 4

Time Domain Diakoptics of TLM Structures

In this chapter, time domain diakoptics using one-port and two-port Johns Matrices are proposed. The techniques are used to cascade microwave guiding structures through convolution.

4.1 Diakoptics Using a One-Port Johns Matrix

Time Domain Diakoptics of TLM structures is the technique that involves breaking down a large TLM structure into small TLM substructures which can be analyzed individually and then reconnected by convolution in the time domain. A so-called Johns Matrix [2] is used to characterize TLM substructures. A one-port Johns Matrix is used when one TLM substructure is connected to another TLM substructure in a single interface. For example, a long waveguide can be broken down into two shorter waveguides. One of the shorter waveguide sections could be analyzed first and characterized by its Johns Matrix. The one-port Johns Matrix is then connected to the other waveguide section by convolution.

The implementation of a full general Johns Matrix involving convolution at all removed branches, as discussed in Chapter 2, is computationally expensive, particularly when the interface contains a large number of such branches. However, as long as only one single mode is present in the interface, a modal convolution involving only a single mode impulse response is required. Note that a valid wideband modal response can only be found if the mode structure function in the interface is independent of frequency, which is the case in homogeneously filled waveguides. In the following, algorithms for connecting two TEM

waveguides and for connecting two TE_{10} waveguides by single mode convolution will be discussed.

4.1.1 Connection of Two TEM Waveguides by Convolution

As shown in Figure 4-1(a), a section of waveguide can be divided into two subsections, TLM1 and TLM2. The Johns Matrix for section TLM2 can be computed independently from the modeling of section TLM1. The overall response of the original structure can be obtained by convolution at interface 1 as shown in Figure 4-1(b). In this case, port 1 of TLM2 is connected to port2 of TLM1; port 2 of TLM2 is terminated with an absorbing boundary. Since there is a single interface between the two TEM waveguides TLM1 and TLM2, a one-port Johns matrix is used to connect them. If the interface contains a large number of TLM link-lines and the solution requires multi-modal convolution, the Johns matrix could be quite large, and the convolution would require considerable computational resources. However, if the field in the interface has a single mode profile then the convolution procedure can be simplified significantly. For instance, if the field at the interface has a constant magnitude across the structure, i.e. the uniform TEM mode situation, then the impulse voltages in all TLM branches across the transverse direction of the structure are the same, so only the impulse response in a single TLM branch needs to be convolved with the excitation signal to obtain the response. Simulation results obtained with the single mode convolution technique are the same (within the numerical round-off error of the computer) as those obtained by direct simulation of the original big structure without partitioning; in most practical situations, the single mode convolution process is also more efficient than the original direct TLM simulation.

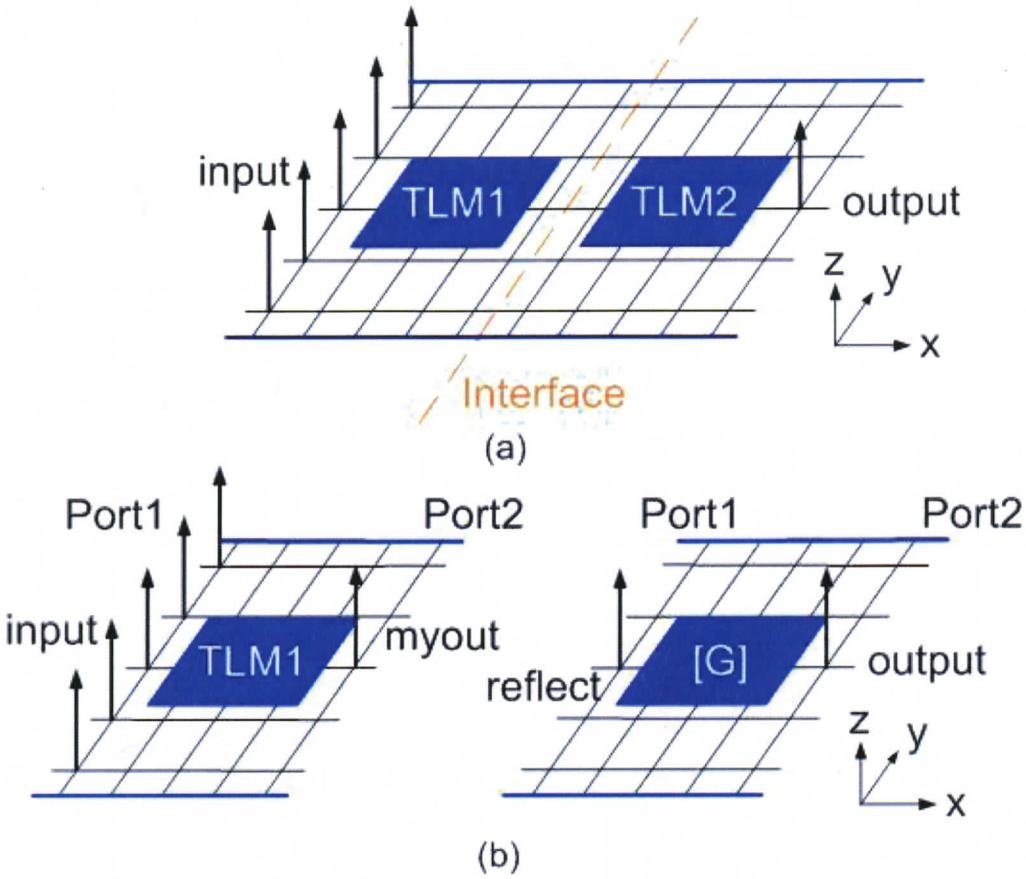


Figure 4-1 (a)Time domain diakoptics of a TLM structure: Dividing a big structure into two small sub-structures TLM1 and TLM2. (b) An illustration of the connection of two TEM substructures: TLM1 and TLM2.

Figure 4-2(a) shows a section of TEM waveguide modeled by a 2D TLM mesh, which is used to compute the Johns Matrix of TLM2 shown in Figure 4-2(b). The single mode two-port Johns matrix of TLM2 is of the form:

$[G] = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix}$ as described in Section 2.4.3 .

To compute the Johns matrix for TLM2, all the voltages in the mesh are initially set to zero, and single unit impulses are injected into all removed branches in port 1 in the interface 1 (in order to excite the TEM mode). All removed branches are terminated with impulse reflection coefficients $\Gamma_i = 0$. G_{11} is then the time domain response picked up at one of the removed branches in port 1, while G_{21} is

the time domain response obtained at one of the removed branches in port 2. In a single-mode TEM case, the reflected voltages are the same in all removed branches in port 1, and similarly for port 2. If the Johns Matrix of TLM2 is symmetric, $G_{11}=G_{22}$ $G_{21}=G_{12}$; otherwise, G_{12} is found by injecting single unit impulses into all removed branches of port 2, and terminating all removed branches with an impulse reflection coefficient $\Gamma_i = 0$. G_{12} is the time domain response picked up at one of the removed branches at port 1. G_{22} is the time domain response picked up in one of the branches in port 2.

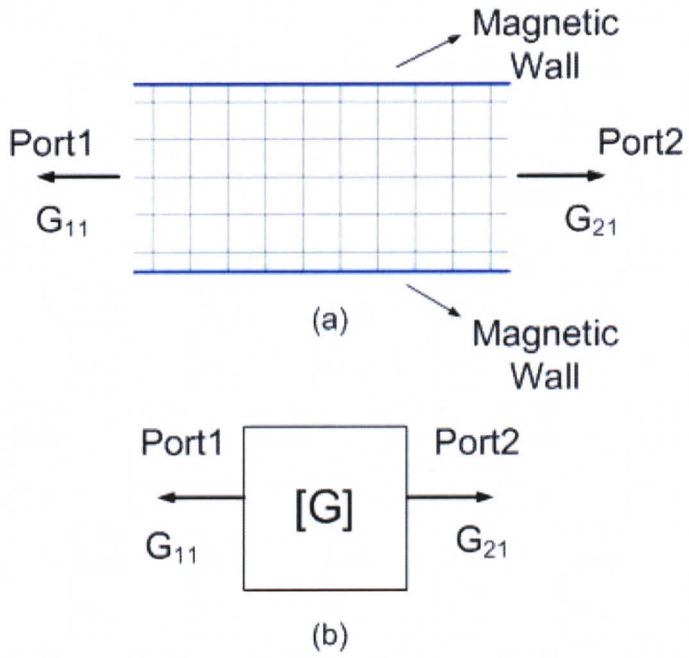


Figure 4-2 (a) A typical TEM structure modeled by a two-dimensional TLM mesh. (b) A block diagram representing the Johns Matrix of substructure, TLM2.

To connect two TEM substructures, i.e. TLM1 and TLM2 from Figure 4-1(b), set initial voltages in the TLM mesh to 0. Inject input unit impulses into all removed branches of TLM1 port1 to obtain output voltages from TLM1 port2 at the interface 1. The reflected voltages at TLM2 port1 are obtained by convolving "*myout*" with G_{11} as shown in Equation 4.1. The output voltages at TLM2 port 2 are obtained by convolving "*myout*" with G_{21} as shown in Equation 4.2.

$$reflect = myout * G_{11} \tag{4.1}$$

$$output = myout * G_{21} \tag{4.2}$$

where "*" denotes numerical convolution.

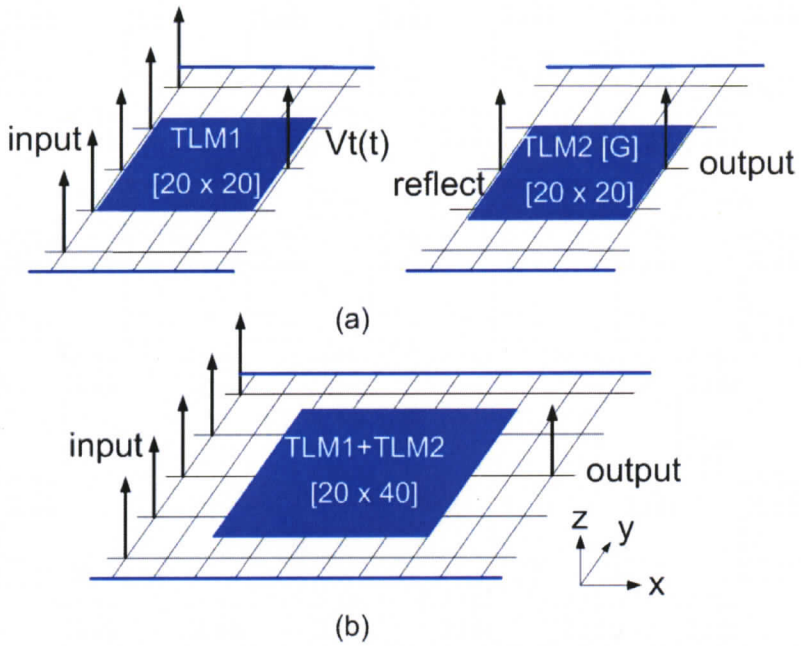


Figure 4-3 An example of connecting two 2D TEM waveguide substructures.

Figure 4-3 shows an example of connecting two 2D TEM waveguide substructures, TLM1 and TLM2. Both substructures are of equal length and discretized by a 20 by 20$\Delta\ell$ TLM mesh. After the Johns Matrix of TLM2 is obtained, TLM1 can be connected to TLM2 using convolution as shown in Figure 4-3 (a). All branches in the interfaces are terminated with their characteristic impedances ($\Gamma_i = 0$). The input of TLM1 is TEM wave represented by a front of unit pulses. The output of TLM1 is Vt(t). In turn, Vt(t) becomes the input of TLM2. The output of TLM2 is obtained by convolving Vt(t) with G_{21} of TLM2. Alternatively, the output of TLM2 can be obtained by directly simulating the whole structure (with size 20 by 40$\Delta\ell$) as depicted in Figure 4-4(b). The convolution and direct simulation approaches produce the same output as shown

in Figure 4-4. The voltage traces overlap and can not be distinguished from one another.

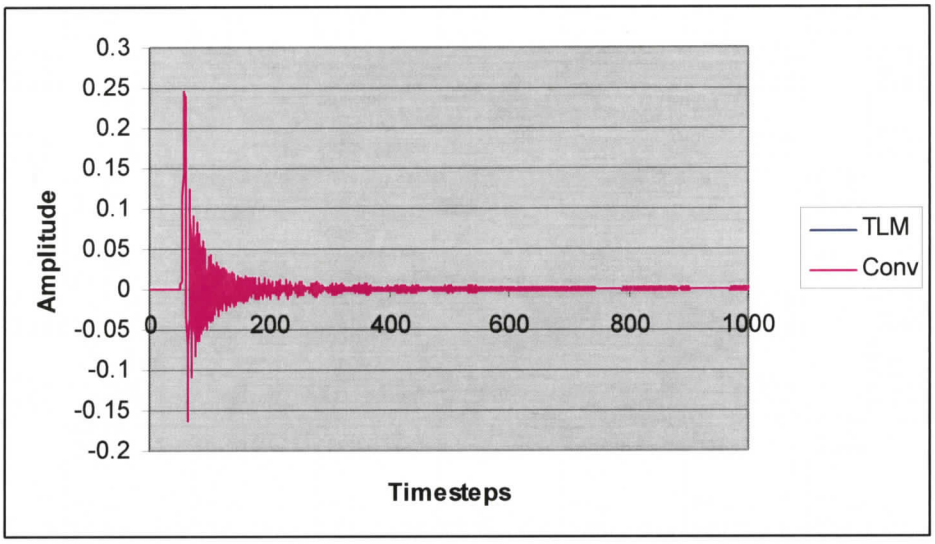


Figure 4-4 Results obtained through convolution (*Conv*) vs. results obtained by simulating the whole structure (*TLM*).

If higher order modes are also present in the connection interfaces, the total field in the interfaces can be thought of as a weighted sum of modal structure functions. Let $V_{tot}(x)$ be the total TLM voltage distribution in the interface between the TEM waveguides, containing the dominant mode and higher order modes with magnitudes A_1 to A_n. The magnitude of the dominant mode A_1 can then be obtained by convolving the transverse field distribution with the dominant modal structure function. In the case of our TEM mode this convolution amounts to computing the average value of the field in the cross-section as shown in Equation (4.3). In other words, A_1 is the spatial average of the total voltage $V_{tot}(x)$.

$$V_{tot}(x) = A_1 + A_2 \cos\frac{\pi x}{a}dx + A_3 \cos\frac{2\pi x}{a}dx + \ldots + A_n \cos\frac{(n-1)\pi x}{a}dx$$

$$\Rightarrow \int_0^a V_{tot}(x)dx = \int_0^a A_1 dx + \int_0^a A_2 \cos\frac{\pi x}{a}dx + \int_0^a A_3 \cos\frac{2\pi x}{a}dx + \ldots \int_0^a A_n \cos\frac{(n-1)\pi x}{a}dx$$

$$\Rightarrow \int_0^a V_{tot}(x)dx = A_1 a$$

Therefore $A_1 = \frac{1}{a}\int_0^a V_{tot}(x)dx$. (4.3)

Figure 4-5 shows an example of how to extract the dominant mode from a general cross-sectional distribution. The sidewalls are magnetic walls, and the top and bottom walls are electric walls. Here, the TEM waveguide cross-section has been divided into five parts A, B, C, D and E with the same length $\frac{a}{5}$. This corresponds to a discretization into five TLM cells of $\Delta l = a/5$ each. The rectangles A – E represent the electric field samples modeled by the impulses in the branches of a TLM mesh with five cells in the cross-section. The total width of the waveguide is a. The average of the field distribution in the cross-section (sampled by the TLM voltages in the branches of cells A to E) is given by Equation (4.4).

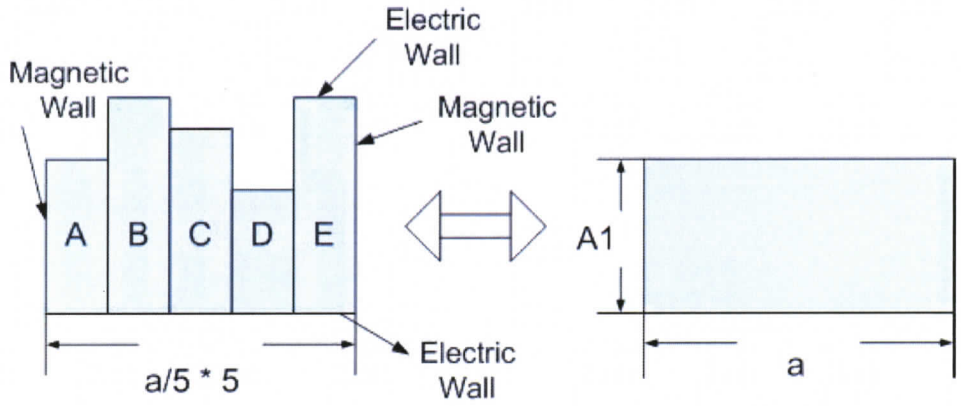


Figure 4-5 An example of extracting the fundamental mode amplitude in a TEM waveguide structure modeled by a TLM mesh with 5 cells in the cross-section.

$$A\times\frac{a}{5}+B\times\frac{a}{5}+C\times\frac{a}{5}+D\times\frac{a}{5}+E\times\frac{a}{5}=A_1\times a$$

Therefore $A_1 = \frac{1}{5}(A+B+C+D+E)$. (4.4)

4.1.2 Connection of Two TE_{10} Waveguides by Convolution

In this section, the extraction of the dominant mode amplitude from a general field distribution in the cross-section of a rectangular waveguide is discussed. A waveguide may contain either an add or even number of removed

branches at the interface as shown in Figure 4-6(a) and Figure 4-6(b), respectively. The dominant TE_{10}-mode features a half-sine distribution of the electric field in port 1. The removed branches at port1 are labelled from branch 1 to $x_{\max}$, where $x_{\max}$ is the total number of removed branches. The middle branch is labeled as mid. For an odd number of removed branches, there is only one middle branch; while for an even number of removed branches, there are two middle branches. If the TLM sample voltage in the middle branch is 1, the voltages in other branches are $\dfrac{\sin\left(\dfrac{x-0.5}{x_{\max}}\pi\right)}{\sin\left(\dfrac{mid-0.5}{x_{\max}}\pi\right)}$, where x is the location of the removed branch. Similarly, at port 2, if the voltage in the middle branch has amplitude A, the voltages in other branches can be found:

for an odd number of branches by: $A\cdot\sin\left(\dfrac{x-0.5}{x_{\max}}\pi\right)$, and

for an even number of branches by: $\dfrac{A}{\sin\left(\dfrac{mid-0.5}{x_{\max}}\pi\right)}\cdot\sin\left(\dfrac{x-0.5}{x_{\max}}\pi\right)$

If we combine the above two formulas, the voltages in other branches for both the odd and even number cases can be expressed by the formula:

$$\frac{A}{\sin\left(\dfrac{mid-0.5}{x_{\max}}\pi\right)}\cdot\sin\left(\frac{x-0.5}{x_{\max}}\pi\right)$$

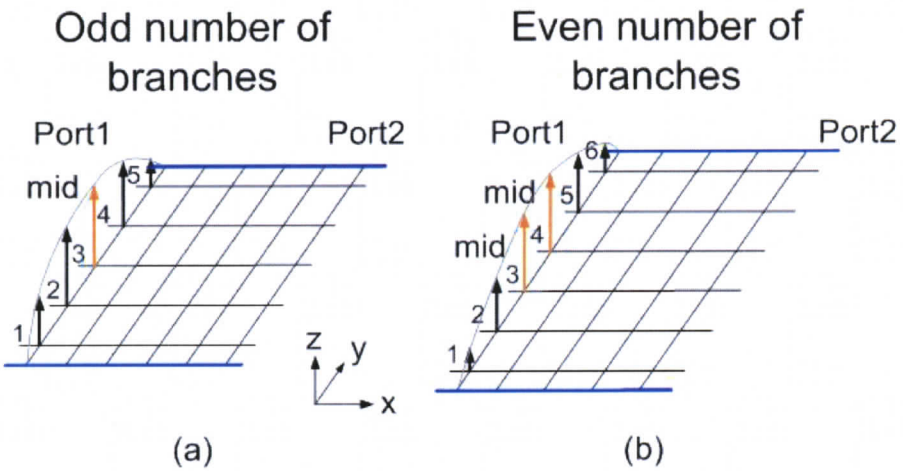


Figure 4-6 (a) Waveguide cross section with an odd number of removed branches and (b) an even number of removed branches.

Similar to a TEM waveguide, a long TE_{10} waveguide can be divided into two TE_{10} subsections, TLM1 and TLM2, as shown in Figure 4-7(a). The response of the complete section can be computed by diakoptics, by finding the modal Johns Matrix for TE_{10} subsection TLM2 and reconnecting it to subsection TLM1 using convolution at the interface of the two subsections. Figure 4-7(b) and Figure 4-7(c) show the Johns Matrix of the TE_{10} subsection TLM2 for an odd and even number of branches at the interface, respectively.

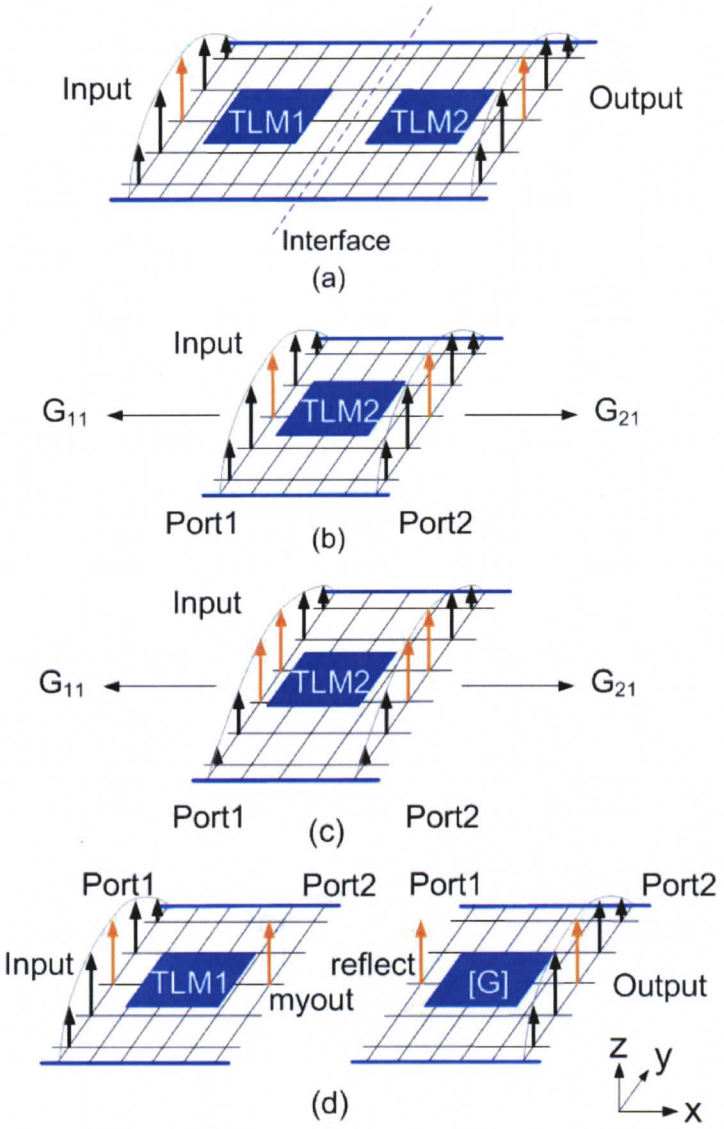


Figure 4-7 (a) Connection of two TE_{10} waveguide substructures. (b) Johns Matrix of TLM2 for an odd number of removed branches at the interface. (c) Johns Matrix of TLM2 for an even number of removed branches at the interface. (d) The connection process of two TE_{10} waveguide substructures.

To find the Johns Matrix of TLM2 for the single mode TE_{10} case, all the initial voltages in the mesh are set to zero. The input of TLM2 is a transverse half-sine distribution in order to excite the TE_{10} mode. All other removed branches are terminated with impulse reflection coefficient $\Gamma_i = 0$. For an interface with an odd number of branches, the voltage in the middle branch is equal to 1, as shown

in Figure 4-7(b). For an interface with an even number of branches in the interface, the voltages in the middle two branches are equal to 1, as shown in Figure 4-7(c). G_{11} is then the time domain response picked up at the middle removed branches in port 1. The reflected voltages in the other removed branches in port 1 can be calculated using the previously given half-sine distribution. G_{21} is the time domain response picked up at the middle removed branches in port 2. The reflected voltages in other removed branches in port 2 can be calculated using the half sine distribution formula. If the Johns matrix of TLM2 is symmetric, $G_{11}=G_{22}$ and $G_{21}=G_{12}$; otherwise, G_{12} is found by injecting voltages with half-sine distribution into all removed branches in port 2, and terminating all removed branches with an impulse reflection coefficient $\Gamma_i = 0$. G_{12} is the time domain response picked up at the middle removed branches at port 1. G_{22} is the time domain response picked up at the middle removed branches in port 2.

Figure 4-7(d) shows the process of connecting two TE_{10} substructures TLM1 and TLM2. First set all the voltages in the TLM mesh to 0, and inject input voltages with a transverse half-sine distribution into all removed branches of TLM1 in port 1. The voltage picked up at the middle removed branch at is called "*myout*". The reflected voltages "*reflect*" emerging from TLM2 are obtained by convolving "*myout*" with G_{11} as shown in Equation 4.5. The total voltages emerging from TLM2 are the result of the convolution of "myout" with G_{21} as shown in Equation 4.6.

$$\text{Reflect} = \text{myout} * G_{11} \tag{4.5}$$

$$\text{Output} = \text{myout} * G_{21} \tag{4.6}$$

where "*" denotes numerical convolution.

An example of connecting two TE_{10} waveguide subsections TLM_1 and TLM_2 by convolution is shown in Figure 4-8(a). TLM_1 and TLM_2 are two TE_{10} waveguide substructures; each discretized by a 20 by $20\Delta\ell$ 2D TLM network. After the TE_{10} Johns Matrix of TLM_2 is obtained, TLM_1 is connected to TLM_2 using convolution. All branches in the interfaces are terminated with characteristic impedances $\Gamma_i = 0$. The input of TLM_1 is a TE_{10} wave with half-sine cross-sectional distribution. The output of TLM_1 Vt(t) becomes the input of

TLM_2. The output of TLM_2 is the convolution of Vt(t) with G_{21} of TLM_2. Figure 4-8(b) shows the equivalent whole structure (with size 20 by 40 $\Delta\ell$) without partitioning. Figure 4-9 compares simulation results obtained through convolution with the results obtained by simulating the whole structure. As expected, the figure shows that the output obtained through convolution is the same as the output obtained by simulating the whole structure without diakoptics.

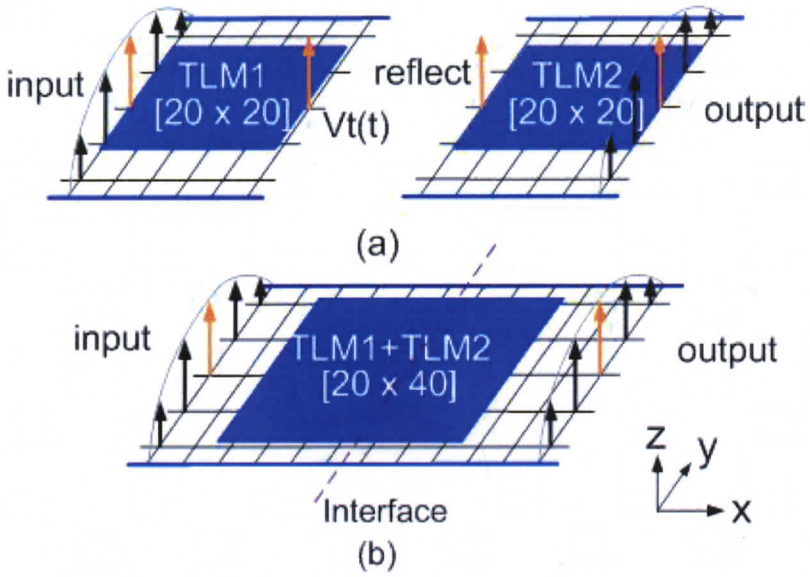


Figure 4-8 (a) Two TE_{10} waveguide substructures. (b) Equivalent structure without partitioning.

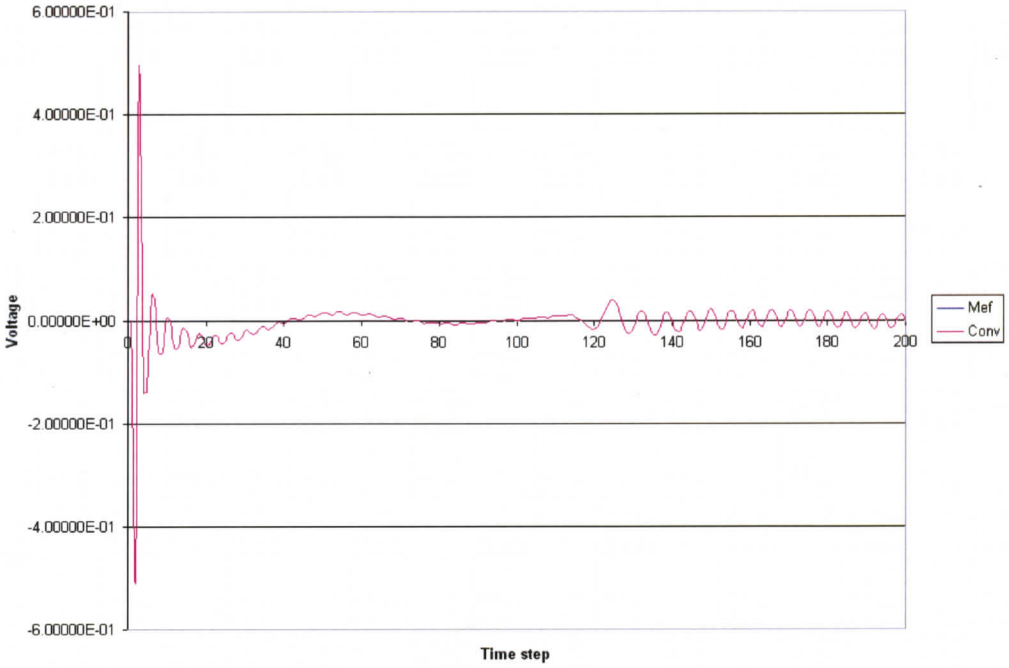


Figure 4-9 Results obtained through convolution (*Conv*) vs. results obtained by simulating the whole structure (*Mef*).

TE_{10} modal extraction:

If higher order modes are also present in the convolution interface between substructures, the dominant mode content must be extracted before convolution. Let $V_{tot}(x)$ be the total voltage distribution in the waveguide cross-section that contains the dominant mode and higher order modes with magnitudes V_1 to V_n. The magnitude of the dominant mode V_1 can be extracted from $V_{tot}(x)$ by means of the trigonometric formula given in Equation (4.7) and (4.8) and the derivation given in Equations (4.9) to (4.11).

$$\sin A \times \sin B = \frac{1}{2}(\cos(A-B) - \cos(A+B)) \tag{4.7}$$

$$\sin^2 x = \frac{1}{2}(1 - \cos 2x) \tag{4.8}$$

$$V_{tot}(x) = V_1 \sin\frac{\pi x}{a} + V_2 \sin\frac{2\pi x}{a} + V_3 \sin\frac{3\pi x}{a} + \ldots + V_n \sin\frac{n\pi x}{a}$$

$$\Rightarrow \int_0^a V_{tot}(x)\cdot\sin\frac{\pi x}{a}dx = \int_0^a V_1 \sin^2\frac{\pi x}{a}dx + \int_0^a V_2 \sin\frac{2\pi x}{a}\sin\frac{\pi x}{a}dx + \int_0^a V_3 \sin\frac{3\pi x}{a}\sin\frac{\pi x}{a}dx + \ldots$$

$$\Rightarrow \int_0^a V_{tot}(x)\cdot\sin\frac{\pi x}{a}dx = \int_0^a V_1 \sin^2\frac{\pi x}{a}dx + \frac{V_2}{2}\int_0^a\left(\cos\frac{\pi x}{a} - \cos\frac{3\pi x}{a}\right)dx + \frac{V_3}{2}\int_0^a\left(\cos\frac{2\pi x}{a} - \cos\frac{4\pi x}{a}\right)dx + \ldots$$

$$\Rightarrow$$

$$\int_0^a V_{tot}(x)\cdot\sin\frac{\pi x}{a}dx = \int_0^a V_1 \sin^2\frac{\pi x}{a}dx + \frac{V_2}{2}\int_0^a\cos\frac{\pi x}{a}dx - \frac{V_2}{2}\int_0^a\cos\frac{3\pi x}{a}dx + \frac{V_3}{2}\int_0^a\cos\frac{2\pi x}{a}dx - \frac{V_3}{2}\int_0^a\cos\frac{4\pi x}{a}dx \ldots$$

$$\Rightarrow V_1 = \frac{\int_0^a V_{tot}(x)\sin\frac{\pi x}{a}dx}{\int_0^a \sin^2\frac{\pi x}{a}dx} \tag{4.9}$$

Since

$$\int_0^a \sin^2\frac{\pi x}{a}dx = \frac{1}{2}\int_0^a\left(1 - \cos\frac{2\pi}{a}x\right)dx$$

$$= \frac{a}{2} - \frac{1}{2}\int_0^a \cos\frac{2\pi}{a}x dx$$

$$= \frac{a}{2} \tag{4.10}$$

Therefore $$V_1 = \frac{\int_0^a V_{tot}\sin\frac{\pi x}{a}dx}{\frac{a}{2}} = \frac{2}{a}\int_0^a V_{tot}\sin\frac{\pi x}{a}dx \tag{4.11}$$

Figure 4-10 shows an example of extracting the fundamental mode from a general cross-sectional distribution. The TE_{10} waveguide cross-section is divided into five parts A, B, C, D and E with the same length $\frac{a}{5}$. This corresponds to a discretization into five TLM cells of $\Delta\ell$ = a/5 each. The total width of the waveguide is a. The dominant mode of the waveguide structure is calculated in Equation (4.12) according to Equation (4.9) and (4.10).

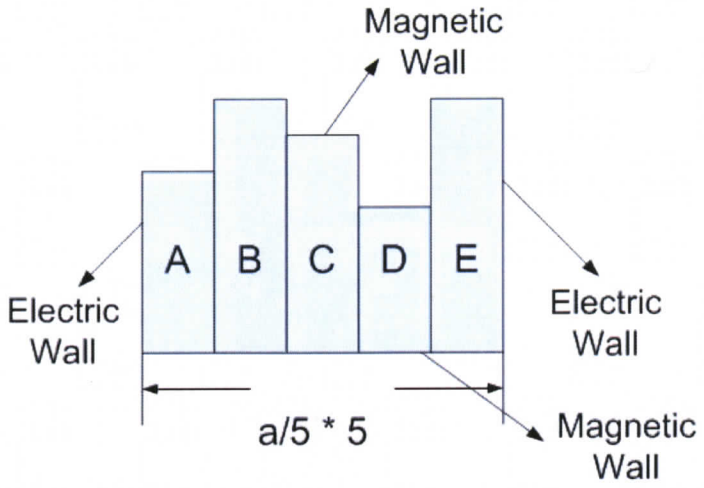


Figure 4-10 Example of extracting the fundamental mode from a TE_{10} waveguide structure.

$$\frac{a}{5}\left(A\cdot\sin\frac{\pi}{a}\cdot\frac{a}{10}+B\cdot\sin\frac{\pi}{a}\cdot\frac{3a}{10}+C\cdot\sin\frac{\pi}{a}\cdot\frac{5a}{10}+D\cdot\sin\frac{\pi}{a}\cdot\frac{7a}{10}+E\cdot\sin\frac{\pi}{a}\cdot\frac{9a}{10}\right)=V_1\cdot\frac{a}{2}$$

$$\frac{1}{5}\left(A\cdot\sin\frac{\pi}{a}\cdot\frac{a}{10}+B\cdot\sin\frac{\pi}{a}\cdot\frac{3a}{10}+C\cdot\sin\frac{\pi}{a}\cdot\frac{5a}{10}+D\cdot\sin\frac{\pi}{a}\cdot\frac{7a}{10}+E\cdot\sin\frac{\pi}{a}\cdot\frac{9a}{10}\right)=V_1\cdot\frac{1}{2}$$

$$V_1=\frac{2}{5}\left(A\cdot\sin\frac{\pi}{a}\cdot\frac{a}{10}+B\cdot\sin\frac{\pi}{a}\cdot\frac{3a}{10}+C\cdot\sin\frac{\pi}{a}\cdot\frac{5a}{10}+D\cdot\sin\frac{\pi}{a}\cdot\frac{7a}{10}+E\cdot\sin\frac{\pi}{a}\cdot\frac{9a}{10}\right) \quad (4.12)$$

4.2 Diakoptics Using a Two-Port Johns Matrix

A one-port Johns matrix structure is used when one structure is terminated by the input port of another, while a two-port Johns Matrix is used when both the reflection and the transmission properties of the second structure are needed. The use of a two-port Johns Matrix is motivated by considering the division of a long waveguide into three smaller waveguide substructures. The middle waveguide section is characterized by a two-port Johns Matrix representing its TE_{10} impulse response. After the Johns matrix of the middle section is obtained, it is connected between the other two TLM substructures using convolution. In this section, novel algorithms for connecting three TEM waveguides as well as three TE_{10} waveguides by convolution are presented, and two examples are discussed.

4.2.1 Cascading of Three TEM Waveguides by Convolution

Figure 4-11(a) shows the division of a long TEM structure into three TEM substructures, TLM1, TLM2 and TLM3. The Johns Matrix representation of the middle TEM substructure TLM2 is shown in Figure 4-11(b). After the Johns matrix of TLM2 is obtained, TLM2 is connected to TLM1 and TLM3 using convolution at port 1 and port 2, respectively. If the fields in the interfaces have a modal structure, the calculation is particularly straightforward. Assuming the simplest case, the fields at the interfaces are those of the uniform TEM mode. Results obtained using convolution should be the same as results obtained from simulating the whole entire structure without partitioning, but depending on the size of the interface, the convolution can be considerably faster than the full simulation.

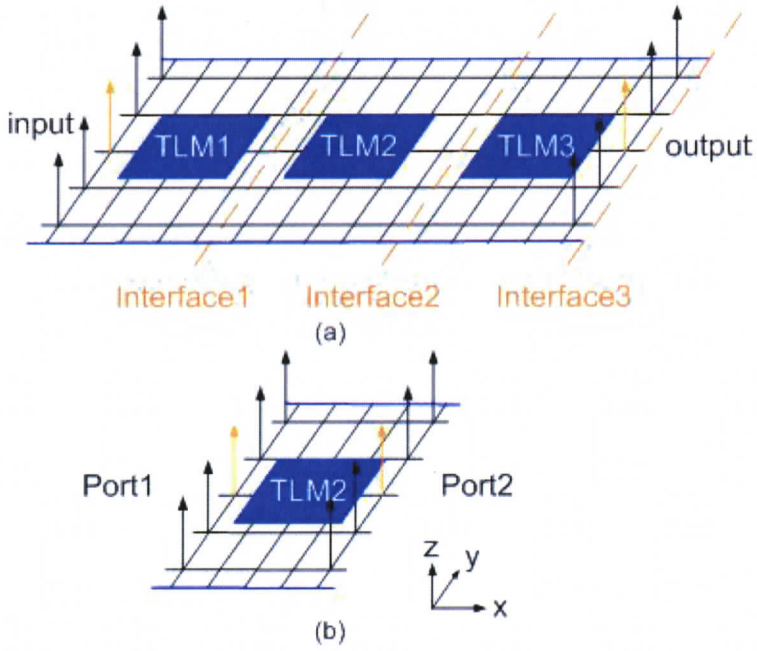


Figure 4-11 (a) Reconnection of three TEM substructures. (b) Finding the two-port Johns matrix of TLM2.

Figure 4-12 shows the process of connecting three TEM substructures TLM1, TLM2 and TLM3 using convolution. The reflection coefficient at the TLM3 boundary interface 3 is J_{11}. Initially, all the voltages in the mesh are set to zero. The *input* to the mesh is applied by injecting single unit impulses into all

removed branches at TLM1 port1. The voltage at the interface 1 of TLM1 is *myout*. The voltage *reflect1* reflected from TLM2 at interface 1 is obtained by convolving *myout* with G_{11}. The voltage *trans1* transmitted through TLM2 at interface 2 obtained by convolving *myout* with G_{21}. When *trans1* reaches interface 3, *reflect2* is reflected from interface 3, where *reflect2* is the convolution of *trans* and J_{11}. When *reflect2* reaches interface 2, part of its voltage is reflected back to TLM3 and the other part is transmitted through TLM2. The reflected voltage *trans1* is obtained by convolving *reflect2* and G_{22}; the transmitted voltage *trans2* is the convolution of *reflect2* and G_{12}. The total reflected voltage at interface 1, *reflect*, is the sum of *reflect1* and *trans2*. The total transmitted voltage *trans* at the interface 2 is the sum of *trans1* and *trans1b*. The *output* could be calculated by the convolving *trans* with G_{21}. This process is described in Equation (4.13).

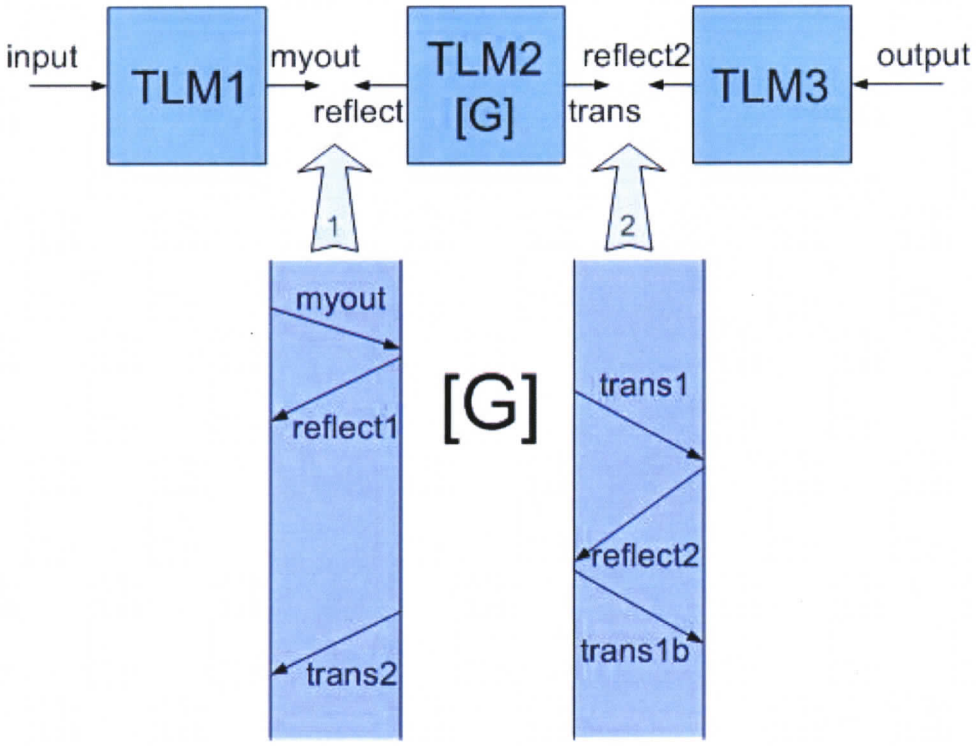


Figure 4-12 Process of connecting three TEM substructures.

$$\begin{cases} reflect1 = myout * G_{11} \\ trans1 = myout * G_{21} \end{cases}$$

$$trans = trans1 + trans1b$$

$$
\begin{aligned}
&reflect2 = trans * J_{11} \\
&\begin{cases} trans1b = reflect2 * G_{22} \\ trans2 = reflect2 * G_{21} \end{cases} \\
&reflect = reflect1 + trans2 \\
&output = trans * G_{21}
\end{aligned} \tag{4.13}
$$

where "*" denotes numerical convolution.

An example of connecting three TEM waveguide subsections by convolution is shown in Figure 4-13. TLM1, TLM2 and TLM3 are three 2D TEM waveguide substructures, each discretized by a $20 \times 20\Delta l$ 2D TLM network. After the TEM Johns Matrix of TLM2 is obtained, TLM2 is connected to TLM1 and TLM3 using convolution. Figure 4-14 compares simulation results obtained through convolution with the results obtained by simulating the whole structure. As expected, the figure shows that the output obtained through convolution is the same as the output obtained by simulating the whole structure (with size $20 \times 60\Delta l$) without partitioning.

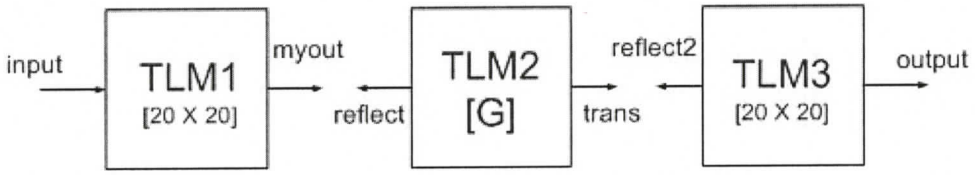


Figure 4-13 Example showing the connection of three TEM waveguide substructures.

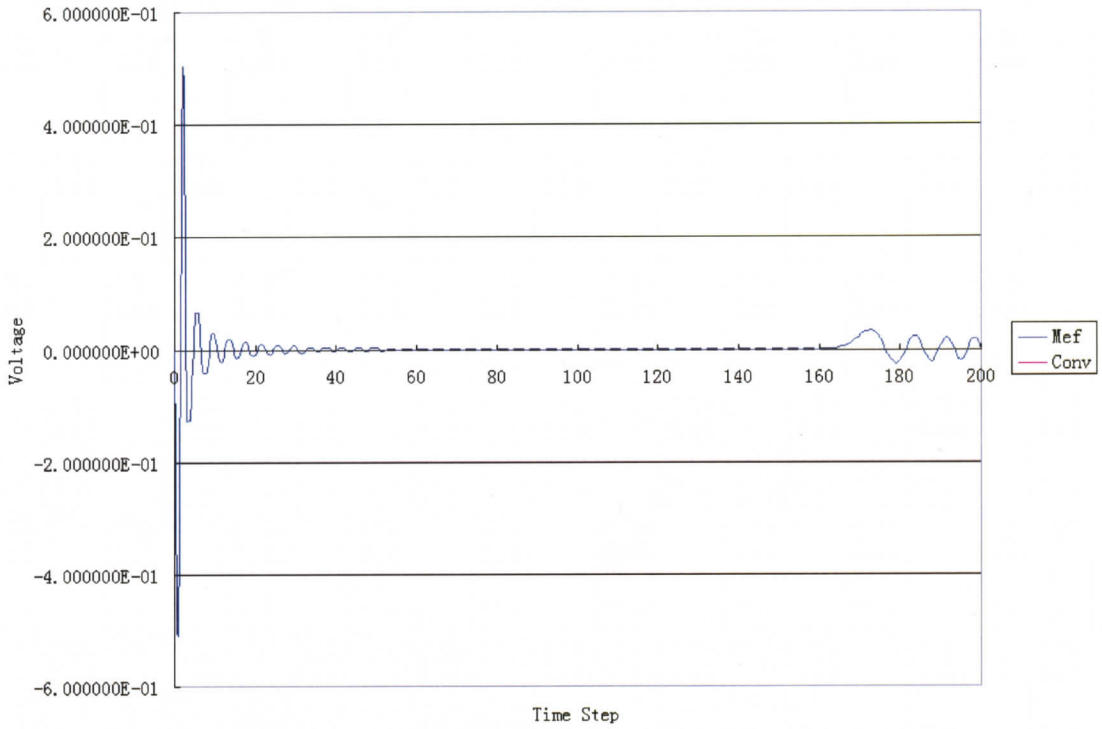


Figure 4-14 Results obtained through convolution (*Conv*) vs. results obtained by simulating the whole structure (*Mef*).

4.2.2 Connection of Three TE_{10} Waveguides by Convolution

Similar to a TEM waveguide, a long TE_{10} waveguide can be divided into three TE_{10} subsections, TLM1, TLM2 and TLM3, as shown in Figure 4-15(a). By using diakoptics, the response of the complete section can be computed by finding the modal Johns Matrix for TE_{10} subsection TLM2 and reconnecting it to the other two TE_{10} subsections TLM1 and TLM3 using convolution at interface 1 and interface 2. Figure 4-15(b) shows the Johns Matrix of TE_{10} subsection TLM2 for an odd number of branches at the interface. The algorithm for the connection of three TE_{10} waveguides by convolution is similar to the algorithm for connecting three TEM waveguides by convolution, which was described previously in section 4.1.2. The exception is that the input has a transverse half sine distribution in order to excite the TE_{10} mode.

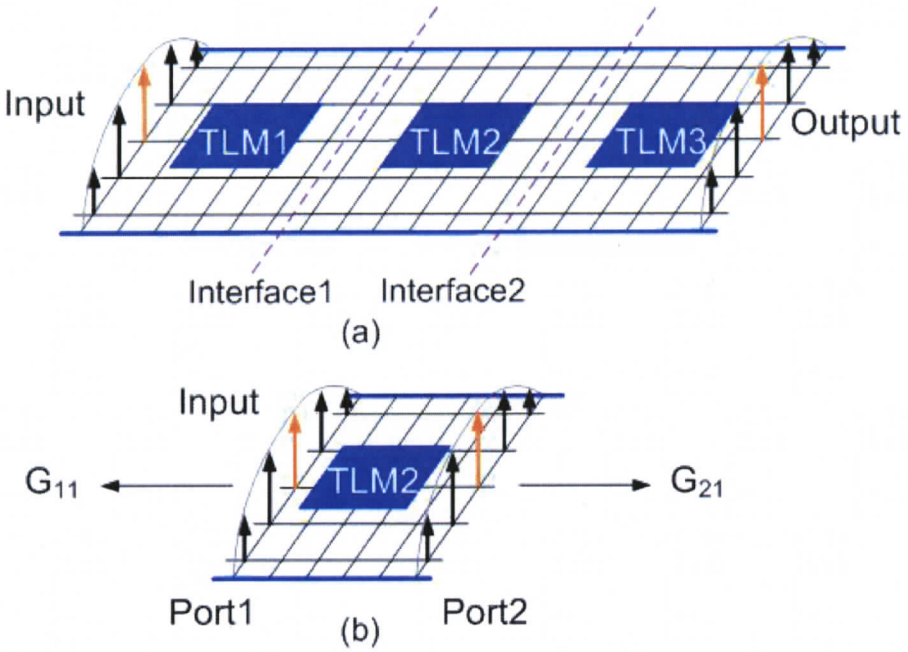


Figure 4-15 (a) Reconnection of three TE_{10} substructures. (b) Finding the two-port Johns matrix of TLM2.

An example of connecting three TE_{10} waveguide subsections by convolution is shown in Figure 4-16(a). TLM1, TLM2 and TLM3 are three TE_{10} waveguide substructures; each discretized by a $20 \times 20\Delta l$ 2D TLM network. After the TE_{10} Johns matrix of TLM2 is obtained, TLM2 is connected to TLM1 and TLM3 using convolution. All branches in the interfaces are terminated with characteristic impedances ($\Gamma_i = 0$). Figure 4-16(b) shows the equivalent whole structure (with size $20 \times 60\Delta l$) without partitioning. Figure 4-17 compares simulation results obtained through convolution with the results obtained by simulating the whole structure. As expected, the figure shows that the output obtained through convolution is the same as the output obtained by simulating the whole structure (with size $20 \times 60\Delta l$) without partitioning.

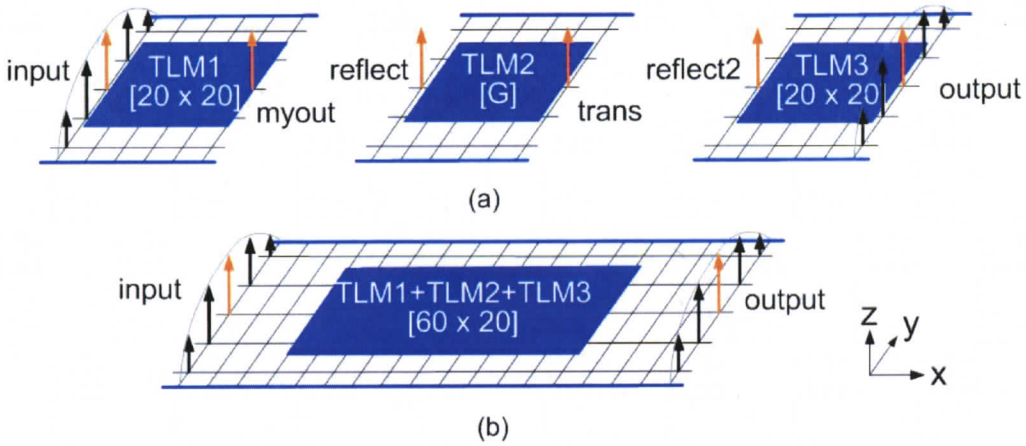


Figure 4-16 Example of reconnecting three TE_{10} substructures.

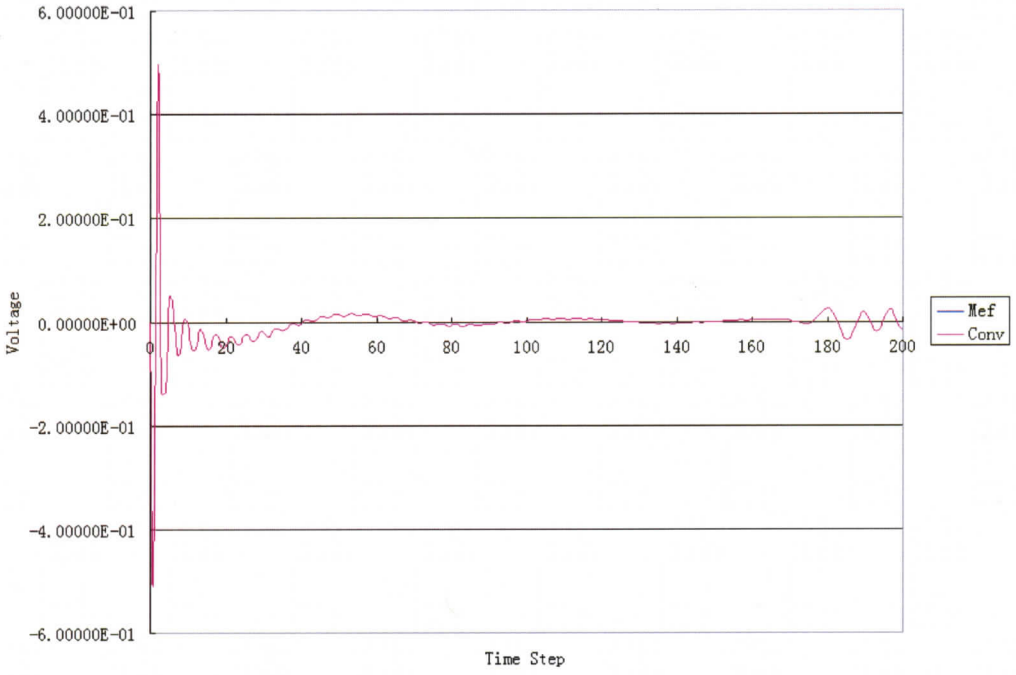


Figure 4-17 Results obtained through convolution (*Conv*) and results obtained by simulating the whole structure (*Mef*).

Chapter 5

Connection between TLM and PSpice

Simulation Program with Integrated Circuit Emphasis (SPICE) is analog and digital circuit simulation software that was developed at the University of California at Berkeley. SPICE is an open source program. PSpice is one of the commercial derivatives of SPICE that runs on personal computers. SPICE and its derivatives are suited for low frequency circuit simulation. In contrast, field-based TLM engines are suited to analyse high frequency components. A co-simulation framework between circuit and field solvers is therefore essential to allow for the analysis of a system comprising both circuits and high frequency components. In this chapter, algorithms for the direct connection between TLM and PSpice will be proposed, followed by the convolution algorithm for connecting one-port and two-port Johns responses to PSpice.

5.1 Direct One-Port Connection Between TLM and Pspice

5.1.1 Background

Park and Russer proposed several techniques [30, 36, 37] for embedding lumped and distributed devices into time-domain transmission line matrix (TLM) models. Du et al. [33, 34] described an analog behavioral model for interconnecting PSpice circuits with time-domain TLM models using two-port S-parameters [38]. So et al. [39, 40] presented a TLM-SPICE interconnection framework based on representing a TLM network as a voltage or current source.

One difficult issue in connecting PSpice with a TLM mesh is defining the location of the interface. The interface can be placed at the node centers, at the cell boundaries located half way between two adjacent nodes or half way between the node center and the cell boundary. If the interface is placed at the node centers, the scattering process is affected since the scattered voltage depends on the voltage reflected from the PSpice circuit. On the other hand, if the interface is placed at the cell boundary, the scattering process remains unaffected since the impulses reflected back to TLM mesh depend on the impulses reflected from the PSpice circuit [39]. If the PSpice circuit does not include active elements, the first two approaches give the same results. Otherwise, the TLM-PSpice interface has to be located in between the node centers and cell boundaries to provide stable and lossless solutions.

Another difficulty in connecting PSpice with a TLM mesh is to define the Thévenin equivalent source V_{TLM} and Thévenin equivalent impedance Z_{TLM} for the TLM mesh. The Thévenin equivalent source V_{TLM} is the instantaneous open circuit voltage in the link lines at the interface that are connected to PSpice. The open circuit voltage of the source at time $k\Delta t$ is shown in Equation (5.1),

$$V_{TLM} = 2 \times V_k^{\,i} = \frac{2}{M} \sum_{m=1}^{M} V_m^{\,i} \tag{5.1}$$

where $V_k^{\,i}$ is the combination of the voltages incident on the PSpice circuit. For example, the voltages of the removed branches at the interface of a 2D TLM mesh (Figure 5.1) are labeled from $V_1^{\,i}$ to $V_m^{\,i}$. Assuming that the characteristic impedances of all link lines are the same, the combination of the voltages incident on the PSpice circuit $V_k^{\,i}$ becomes the average of the M stack voltages $V_1^{\,i}$ to $V_m^{\,i}$. The Thévenin equivalent impedance Z_{TLM} for the TLM mesh becomes the shunt combination of the link line impedances as shown in Equation (5.2) [39] .

$$Z_{TLM} = \frac{Z_{lm}}{M} \tag{5.2}$$

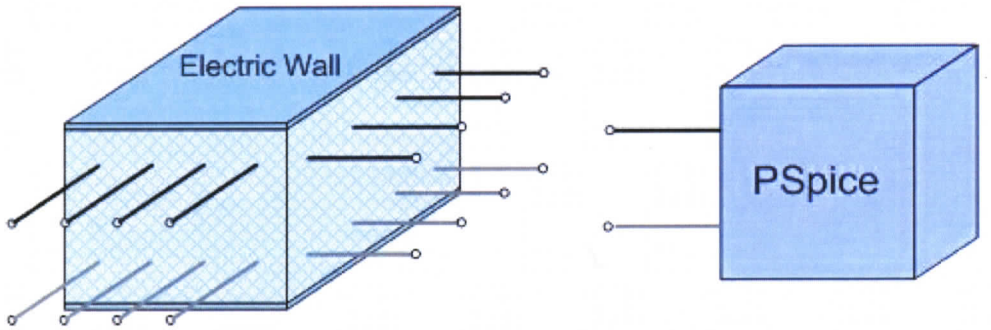


Figure 5-1 Example of connecting a 2D TLM mesh with a Pspice circuit model.

With the Thévenin equivalent sources V_{TLM} and the Thévenin equivalent impedance Z_{TLM} defined, the PSpice and TLM mesh interconnection reduces to the circuit shown in Figure 5-2.

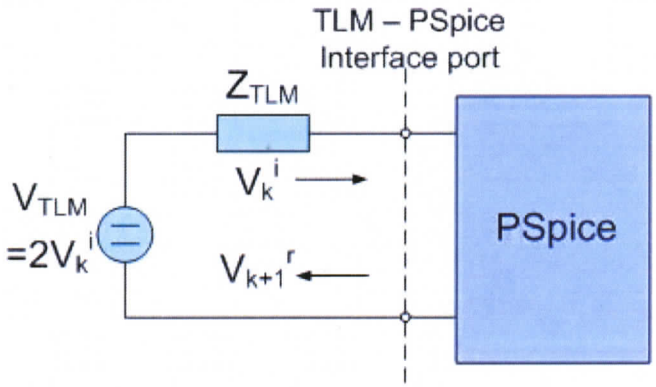


Figure 5-2 Equivalent circuit of 2D TLM mesh and Pspice connection using Thévenin equivalent source and Thévenin equivalent impedance.

The calculation of the reflected voltage V_{k+1}^{r} from PSpice at the TLM-PSpice interface requires further consideration. The reflected voltage from PSpice depends on the location of the TLM-PSpice boundary. The TLM-PSpice boundary may be placed at one of three locations: at the node center, at the cell boundary, or half-way between the node center and the cell boundary. The choices of boundary location are called the node implementation, the boundary implementation and the average implementation, respectively shown in Figure 5-3

[39]. The calculation of the reflected voltage gives the same results and is independent of the choice of boundary location, if there is no resonant element in the PSpice circuit. Otherwise, placing the TLM-Pspice boundary half-way between the node center and the cell boundary gives the most stable and lossless results [40].

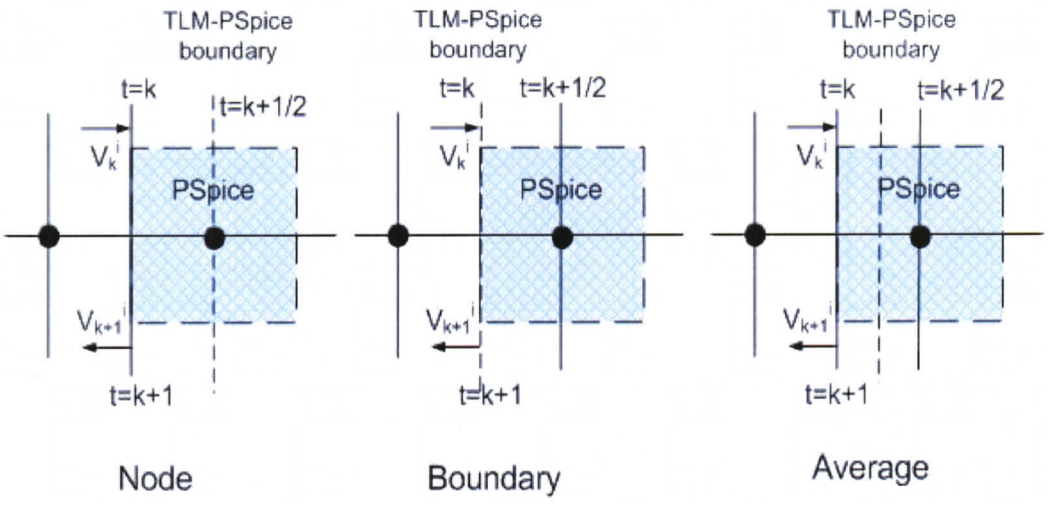


Figure 5-3 Three possible locations of the TLM-PSpice boundary: at the node center, at the cell boundary, and half way between the node center and the cell boundary.

For the node implementation, the reflected voltage V_{k+1}^{r} from PSpice is calculated by Equation (5.3):

$$V_{k+1}{}^{r} = PSpice_{k+\frac{1}{2}}^{k+\frac{1}{2}}(V_k{}^{i}) - V_k{}^{i} \tag{5.3}$$

where $PSpice_{k+\frac{1}{2}}^{k+\frac{1}{2}}(V_k{}^{i})$ is the instantaneous total voltage computed by PSpice at time $t = k + \frac{1}{2}$. $V_k{}^{i}$ is the incident voltage at the cell boundary at time $t = k$. V_{k+1}^{r} is the reflected voltage to the TLM mesh from the PSpice circuit at time $t = k + 1$.

For the boundary implementation, the reflected voltage V_{k+1}^{r} from PSpice is calculated by Equation (5.4):

$$V_{k+1}{}^{r} = PSpice_{k}^{k+1}(V_k{}^{i}) - V_k{}^{i} \tag{5.4}$$

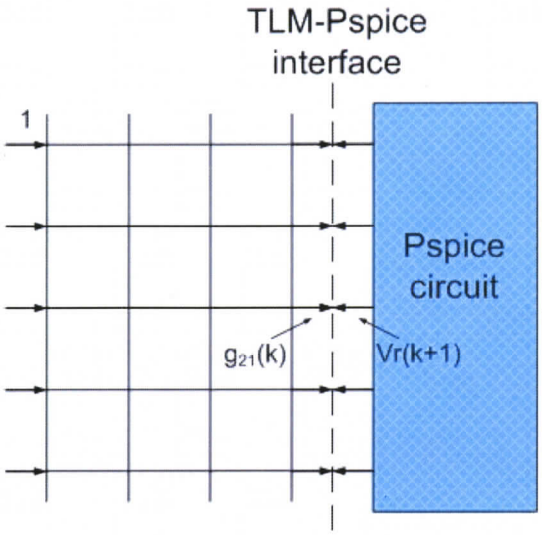


Figure 5-4 The connection of a PSpice circuit with a parallel plate waveguide modeled by a 2D TLM shunt mesh.

$$V_s(k) = 2 \times g_{21}(k) \tag{5.6}$$

$$R_s = \frac{Z_l}{n} = \frac{Z_o\sqrt{2}}{n} \tag{5.7}$$

After the Thévenin equivalent voltage source $V_s(k)$ and the Thévenin equivalent impedance R_s are obtained, the direct TLM and PSpice connection is simplified to the circuit shown in Figure 5-5. The PSpice engine is invoked by executing the command called psp_cmd.exe. PSpice does one time-step calculation, and the resulting voltage V_{k+1}^r is applied to the TLM mesh at time $t = (k+1)\Delta t$. V_{k+1}^r is calculated by Equation (5.8):

$$V_{k+1}{}^r = \frac{1}{2}(PSpice_k^k(V_k{}^i) + PSpice_k^{k+1}(V_k{}^i)) - g_{21}(k) \tag{5.8}$$

The incident voltage V_k^i arrives at the TLM-PSpice interface at time $t = k$ and leaves at time $t = k+1$. $PSpice_k^{k+1}(V_k^i)$ is the difference between the voltages calculated by PSpice at $t = k+1$ and $t = k$.

For the average implementation, the voltage V_{k+1}^r reflected from PSpice is calculated by Equation (5.5):

$$V_{k+1}{}^r = \frac{1}{2}(PSpice_k^k(V_k{}^i) + PSpice_k^{k+1}(V_k{}^i)) - V_k{}^i \tag{5.5}$$

where $PSpice_k^k(V_k^i)$ is the voltage at the TLM-PSpice interface port at time $t = 0k$. $PSpice_k^{k+1}(V_k^i)$ is the difference between the voltages calculated by PSpice at $t = k+1$ and $t = k$. The average implementation is the combination of the node implementation and boundary implementation. If the PSpice circuit includes resonant structures, the average implementation gives the most stable implementation under all tested situations [39]. The average implementation is used to calculate the reflected voltage in all remaining examples. $PSpice_k^k(V_k^i)$ and $PSpice_k^{k+1}(V_k^i)$ terms can be obtained from the PSpice output file, filename.out.

5.1.2 Test of Implementation 1

In this section, a method to create a direct one-port connection between a TLM and PSpice is proposed, and followed by applications to a detailed example. Figure 5-4 shows the connection of a PSpice circuit with a parallel plate waveguide that is modeled by a 2D TLM shunt mesh.

Unit impulses are injected as input into the removed branches of the TLM mesh at the beginning of the simulation, t=0. The parallel plate waveguide supports a TEM wave, hence all the voltage pulses at the TLM-PSpice interface plane are the same. The voltage impulse $g_{21}(k)$ at one of the removed branches in the TLM-PSpice interface is picked up at time $t = k\Delta t$. The Thévenin equivalent voltage source $V_s(k)$ and the Thévenin equivalent impedance R_s are calculated by Equations (5.6) and (5.7).

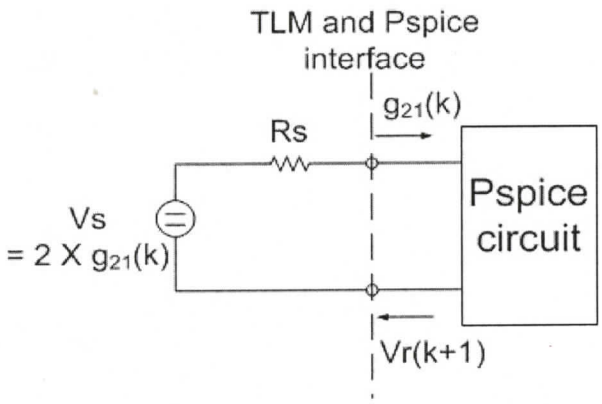


Figure 5-5 Equivalent circuit of the connection of PSpice circuit and a parallel plate waveguide modeled by a 2D TLM mesh.

Figure 5-6 shows an example of a direct one-port connection between a parallel plate waveguide modeled by a 2D TLM shunt mesh and a PSpice circuit. The parallel plate waveguide has a cross-section of $20mm \times 20mm$ and a mesh size of 1mm in the x and y direction. *Vs* and *Rs* are the Thévenin equivalent source and Thévenin equivalent impedance of the TLM mesh, respectively. The propagation delay for 1 cell (1mm) is Δt:

$$\Delta t = \frac{\Delta l}{\sqrt{2} \cdot c} \tag{5.9}$$

where c is the speed of light ($2.99792X10^8$ m/s); and Δl is the mesh size (1mm). Therefore, the propagation delay for 1 cell is 2.35865 ps, which defines the time-step of the PSpice simulation. After each time-step, the PSpice results are used to calculate the reflected voltage V_{k+1}^r that is returned to the TLM mesh.

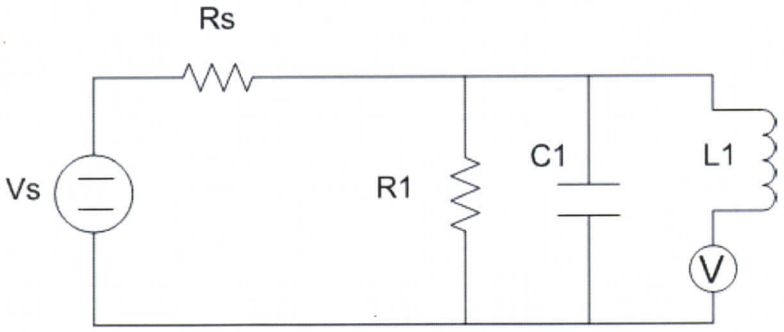


Figure 5-6 Direct connection of a parallel plate waveguide with a PSpice circuit consisting of a resistor R_1, a capacitor C_1 and an inductor L_1 connected in shunt.

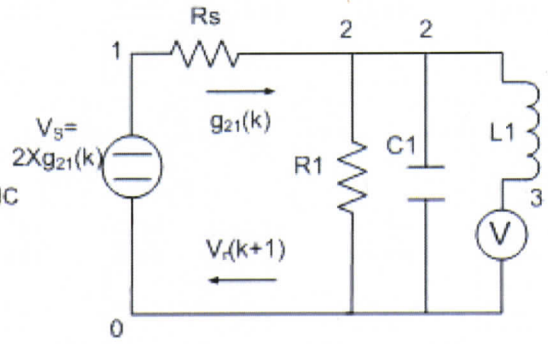
The PSpice circuit consists of a shunt connected resistor $R_1 = 5\Omega$, capacitor $C_1 = 20\,pF$ and inductor $L_1 = 20\,pH$. The circuit netlist is shown in Figure 5-7.

```
*
**
Vs  1  0 DC   0
*
Rs   1    2    26.6579
R1   2    0    5
C1   2    0    20p  IC= 0
L1    2    3    20p  IC= 0
V    3    0    0
*
* ANALYSIS
.TRAN    2.35865ps  2.35865ps 0us 2.35865ps UIC

* VIEW RESULTS
.PRINT    TRAN     V(1) V(2)  I(V)
.probe
.END
```

Initial voltage condition for capacitor C1.

Initial current condition for inductor L1.



(a) (b)

Figure 5-7 Direct TLM and PSpice connection: (a) Netlist file for the PSpice engine. (b) Equivalent circuit.

The locations of each component in the PSpice circuit are labeled by node number 1 to 3. *Vs* is the DC voltage source with initial value 0. The value of *Vs* is updated at each time-step based on the calculated voltage $g_{21}(k)$ from the TLM mesh at time $t = k$ according to Equation (5.6). The Thévenin equivalent impedance *Rs* is calculated using Equation (5.7). In this example, there are

twenty removed branches at the TLM and PSpice interface. Therefore, $n = 20$ and Rs is:

$$Rs = \frac{Zl}{n} = \frac{Zo\sqrt{2}}{n} \approx \frac{377\Omega \times \sqrt{2}}{20} \approx 26.6579\Omega$$

The initial conditions for the capacitor and the inductor need to be updated at each time-step too. The voltage across the capacitor changes with time, so changing the initial voltage condition for the capacitor can be done by updating the voltage across the capacitor $IC = ?$ value. The current through the inductor changes with time, so changing the initial current condition for the inductor can be done by updating the current through the inductor $IC = ?$ value. Another voltage source set to 0V DC is connected to the inductor in series. Therefore the current passing through the inductor is also the current passing through the DC voltage source. The transient analysis performed by PSpice is controlled by the line starting with *.TRAN* in the netlist. That line has the following format:

.TRAN prt_stp t_max prt_dly max_stp UIC

where *prt_stp* is the time-step used for exporting information to the output file.

t_max is the time at which the simulation will be ended.

prt_dly is the print delay. Only the data in the time interval of interest are printed.

Max_stp is the maximum step size that PSpice is allowed to take during simulation.

UIC informs PSpice to use initial condition [41].

Since only the results at the beginning and the end of each simulation time-step are of interest, the values for the parameters *prt_stp*, *t_max* and *max_stp* are identical and equal to $1\Delta t = 2.35865ps$. If *prt_dly* is set to $0\mu s$, all the data from the beginning of the simulation to the end of the simulation are printed.

The line starting with *.print* determines the voltages and currents that are stored in the PSpice output file, *filename.out*. The initial conditions of the capacitor and the inductor are updated at each time-step using results from the previous time-step. V(2) is used to update the initial voltage condition for the capacitor. I(3) is used to update the initial current condition for the inductor. Node

2 and node 0 are at the TLM and PSpice interface. The reflected voltage $V^r(k+1)$ to be returned to the TLM mesh from PSpice is calculated by Equation (5.8), where $g_{21}(k)$ is the voltage going from the TLM mesh to PSpice; $PSPICE_k^k(g_{21}(k))$ is the value of V(2) at the beginning of each simulation; $PSPICE_k^{k+1}(g_{21}(k))$ is the value of V(2) at the end of each simulation. $V^r(k+1)$ is injected into each removed branch of the TLM mesh at the TLM-PSpice interface. The line with .END defines the end of the PSpice netlist file. The PSpice netlist file is created in notepad and saved as *filename.cir* (in this example, it is saved as input.cir). The PSpice engine is invoked by the command line:

psp_cmd input.cir.

where input.cir is the filename of the netlist.

The initial condition values that PSpice calculates are stored in the PSpice output file (*input.out* in this example). The first part of the output file is the original PSpice circuit file; the second part contains the results of the transient analysis results as follows:

TIME	V(1)	V(2)	I(V)
0.000E+00	-1.051E-05	5.334E-06	-3.067E-07
2.359E-12	-1.051E-05	5.140E-06	3.116E-07

The four columns show the simulation time, the voltage across the Thévenin equivalent source V(1), the voltage across the capacitor V(2), and the current through the inductor I(V) at the start and the end of the simulation, respectively. For the results shown, the voltages across the capacitor and inductor at the end of PSpice one time-step simulation are 5.140E-06 V and 3.116E-07 A, respectively. They are the initial voltages for the capacitor and inductor at the next time-step. The voltage reflected back to the TLM mesh is the average of V(2) at the start and the end of the simulation minus the incoming voltage $g_{21}(k)$ from the TLM mesh according to Equation (5.8).

The flow chart shown in Figure 5-8 summarizes the direct one-port connection between PSpice and a parallel plate waveguide. In the initialization

stage, all the voltages in the TLM mesh are set to zero. When voltage impulses reach node centers, scattering occurs at the scattering stage. A PSpice netlist file describes the circuit and components being analyzed, and sets the simulation time and the variables to be computed. The input netlist file is initially created in notepad and saved as input.cir. The initial voltage of the Thévenin equivalent source *Vs* is set to zero. The value of *Vs*, and the initial conditions (IC) of the capacitor and the inductor must be updated at each time-step. Since it is not possible to read the netlist and modify its contents at the same time, input.cir is copied to input2.cir. The locations of the values to be updated are determined while reading input2.cir. After values are updated, input2.cir is used to overwrite the original netlist input.cir. The next time-step is simulated by PSpice using the following command at time $t = k\Delta t$:

psp_cmd input.cir

The PSpice output file input.out is generated after the simulation of the time-step. The voltage across the capacitor and the current through the inductor at the end of the simulation are retrieved from the output file. Those values are the initial conditions for the capacitor and inductor at the next time-step, and are used to update the input circuit file input.cir. From the netlist file, the voltage at the TLM-PSpice interface is obtained to calculate the voltage $V^r(k+1)$ going back to the TLM mesh at time $t = (k+1)\Delta t$ as shown in Equation (5.8). $V^r(k+1)$ is also the voltage used to update the TLM boundary condition. In the connection stage, voltage pulses from one node are exchanged with the voltage pulses arriving from its adjacent nodes. The process repeats itself starting at the scattering stage, until the total simulation time t_{total} is exceeded.

To validate the proposed algorithm, the reflection coefficient S_{11} is calculated at a node in the TLM mesh. Figure 5-9 compares the S_{11} magnitude $|S_{11}|$ and phase $\angle S_{11}$ obtained from the proposed algorithm against a MEFiSTo-2D simulation. The figure shows that the S_{11} results obtained with the algorithm and results obtained with MEFiSTo-2D are in excellent agreement.

In this example, resistor R_1, capacitor C_1 and inductor L_1 form a resonant circuit. The resonant frequency of the circuit is given by Equation (5.10):

$$f_o = \frac{1}{2\pi\sqrt{LC}} \tag{5.10}$$

With $C_1 = 20pF$ and $L_1 = 20pH$, the resonant frequency of the circuit is 7.9577 GHz. The theoretical resonant frequency is close to the simulated resonant frequency of 7.866 GHz in Figure 5-9 with a 1.15% difference. This difference is due to the coarseness of the TLM mesh. The TLM mesh size should be decreased to reduce the error.

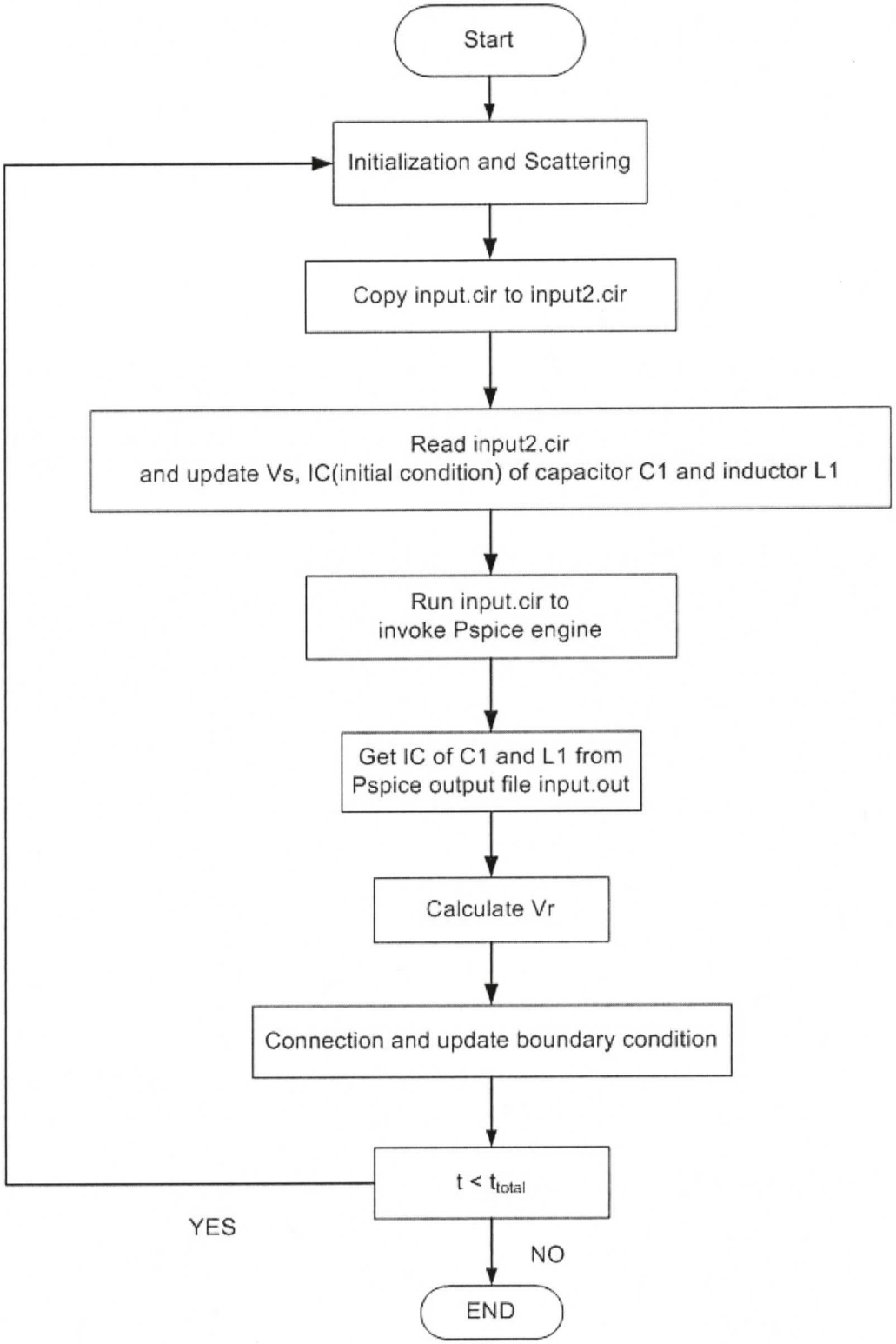


Figure 5-8 Flow chart illustrating the process of connecting a parallel plate waveguide with a PSpice circuit.

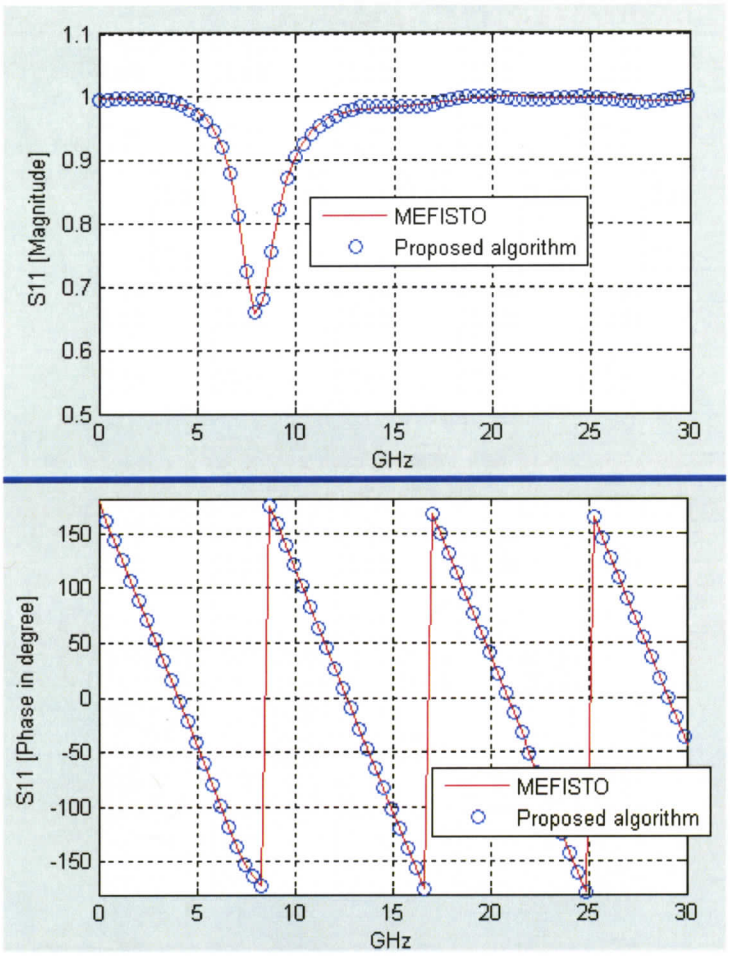


Figure 5-9 Simulation results obtained by direct connection of TLM and PSpice circuit labelled "Proposed algorithm" and results from MEFiSTo-2D labelled "MEFISTO".

5.1.3 Test of Implementation 2

In this section, an example of a direct one-port connection between PSpice and two TEM parallel plate waveguides is presented as shown in Figure 5-10. Both TLM_1 and TLM_2 in Figure 5-10 are TEM parallel plate waveguides with of size $20mm \times 20mm$. The 2D TLM structure is modeled in MEFiSTo-2D with mesh size $1mm$ as shown in Figure 5-11(a). Figure 5-11(b) is the reference structure featuring an absorbing boundary on the right side with reflection coefficient $\Gamma = 0$. This reference structure is used to calculate S parameter S_{11} in

phase $\angle S_{11}$ and in magnitude $|S_{11}|$ at a specific node. The PSpice circuit consists of shunt connected $R_1 = 5\Omega$, $C_1 = 20pF$ and $L_1 = 20pH$.

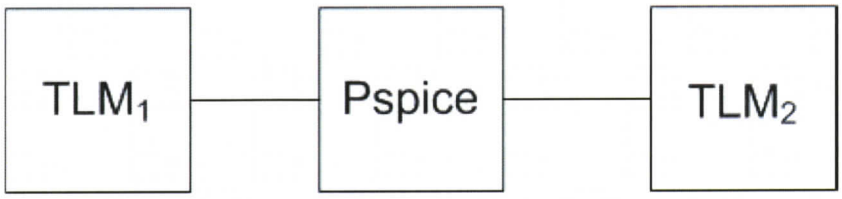


Figure 5-10 Direct one-port connection between PSpice and two TEM parallel plate waveguides TLM_1 and TLM_2.

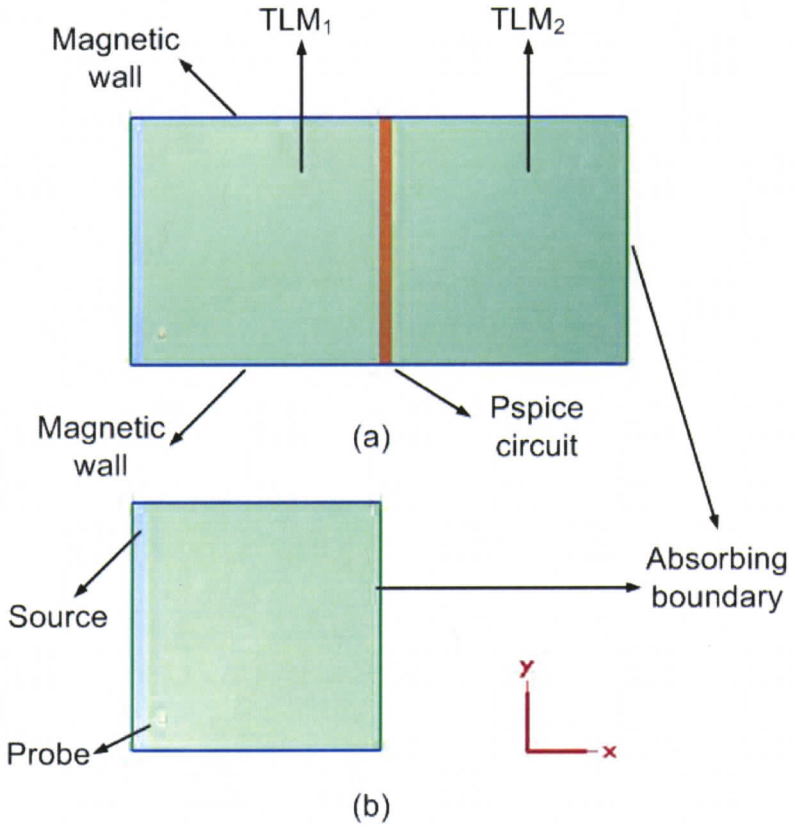


Figure 5-11 (a) Connection of TEM parallel plate waveguides TLM_1, TLM_2 and PSpice circuit modeled by MEFiSTo-2D; (b) Reference structure used to calculate the reflection coefficient S_{11}.

The PSpice input circuit which is passed to the PSpice engine has to be created first. Since TLM1 and TLM2 has size of $20mm \times 20mm$, with mesh size $1mm$, there are twenty removed branches in the TLM-PSpice interface. The Thévenin equivalent impedances for both TLM_1 and TLM_2, RE1 and RE2, respectively, can be calculated using Equation (5.7) with $n = 20$, so $RE1 = RE2 = 26.6579\Omega$. Figure 5-12 shows the equivalent PSpice circuit for this direct one-port connection between PSpice and two TEM parallel plate waveguides. Assume PSpice takes one cell of space. The propagation delay for 1 cell (1mm) Δt is 2.35865ps according to Equation (5.9). The PSpice circuit is saved to a text file, input.cir, and it is passed to the PSpice engine. PSpice does one time-step calculation and returns the reflected voltage back to TLM_1 and TLM_2 at the next time-step.

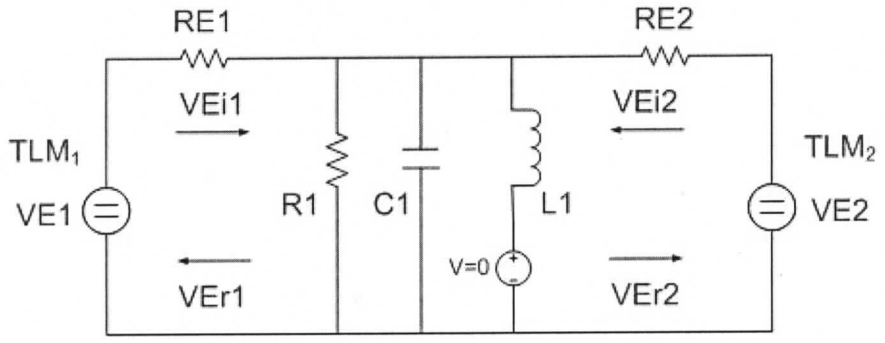


Figure 5-12 The equivalent circuit of the interconnection of TLM_1, TLM_2 and a PSpice circuit.

The connection framework of this example is similar to the previous example; see the flow chart in Figure 5-8 for details. At the initialization stage, all the voltages and initial conditions for the capacitor and the inductor are set to zero. At the scattering stage, both TLM_1 and TLM_2 do the scattering at the same time. The voltages incident from TLM_1 and TLM_2 on the PSpice circuit at the TLM-PSpice interface are called $VEi1$ and $VEi2$, respectively. $VEi1$ and $VEi2$ are used to calculate the Thévenin equivalent sources $VE1$, $VE2$ for TLM_1 and TLM_2, respectively at time $t = k\Delta t$. The PSpice circuit file must be updated at each time-step by changing the initial condition of the capacitor, inductor and $VE1$ and $VE2$. The initial conditions for the capacitor and inductor for the next

time-step are read from the PSpice output file input.out. The reflected voltages $VEr1$ and $VEr2$ back to TLM_1 and TLM_2 are calculated using Equation (5.5) based on the voltages incident on the PSpice circuit $VEi1$ and $VEi2$, respectively. At the connection and boundary condition stage, voltage pulses from one node are exchanged with those emerging from neighbor nodes in TLM_1 and TLM_2. At last, if the time t is smaller than the total simulation time t_{total}, the simulation needs to go back to scattering stage; otherwise, the simulation is terminated.

Figure 5-13 depicts the return loss computed with the proposed algorithm and with MEFiSTo-2D; the results agree very well with each other.

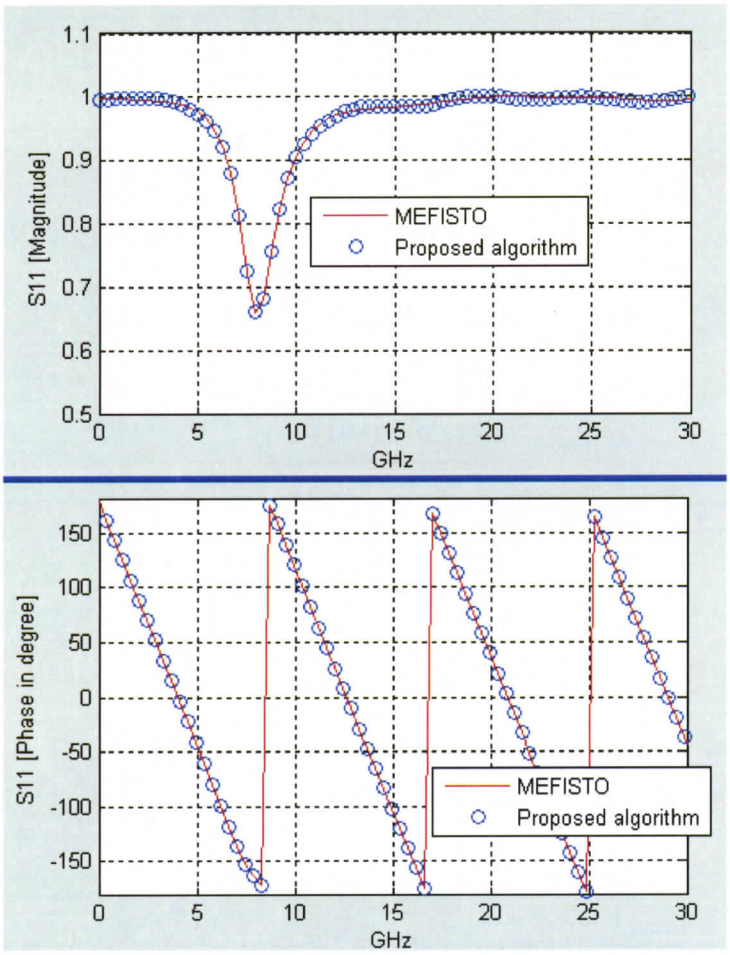


Figure 5-13 Simulation results obtained by direct connection of TLM_1, TLM_2 and PSpice circuit labelled "Proposed algorithm" and results from MEFiSTo labelled "MEFISTO".

5.2 Connection of One-Port Johns Response to PSpice by Convolution

In the previous section, algorithms of direct connection between one-port parallel plate waveguides and a PSpice circuit have been introduced. To reduce the total simulation time, diakoptics could be applied as well. In the section 4.1.1, a large TEM waveguide was divided into two subsections, and they were reconnected again using convolution. The same methodology can also be applied if one of the TLM subsections is substituted by a PSpice circuit. In the first step, the TLM subsection is characterized by its one-port Johns response. Then, the TLM subsection is connected with the PSpice circuit by convolution with the pre-calculated one-port Johns response.

In this section, an example of connecting the one-port Johns response to PSpice by convolution will be described. Let us revisit the example in the section 5.1.2 where we have described a direct connection between TLM and PSpice. The connection between TLM and PSpice by convolution is shown in Figure 5-14. Both TLM_1 and TLM_2 are TEM parallel plate waveguides with size $20mm \times 20mm$ and mesh size $1mm$. The 2D TLM structure which is modeled in MEFiSTo-2D is shown in Figure 5-11. The PSpice circuit consists of $R_1 = 5\Omega$, $C_1 = 20pF$ and $L_1 = 20pH$, which are connected in shunt.

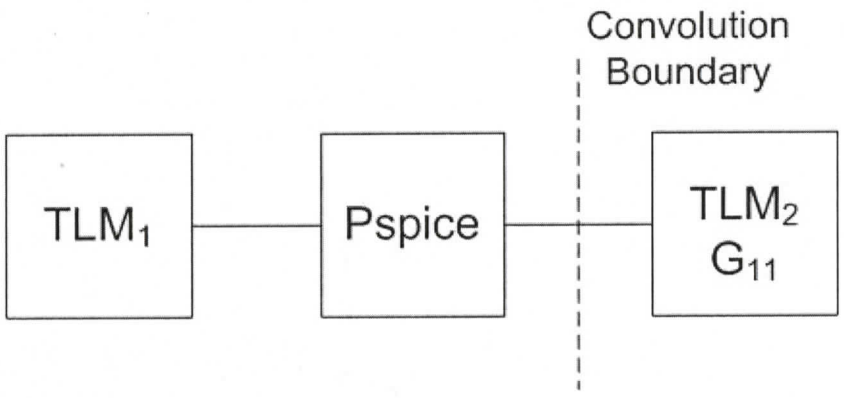


Figure 5-14 The interconnection of TLM_1, PSpice and TLM_2 by convolution.

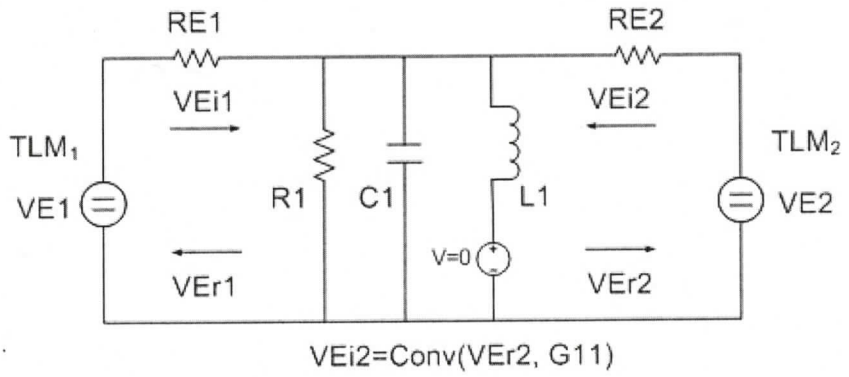


Figure 5-15 The equivalent circuit of the interconnection of TLM_1, PSpice and TLM_2.

Figure 5-15 shows the equivalent circuit of the connection of TLM_1, TLM_2 and PSpice. The PSpice circuit consists of shunt-connected $R_1 = 5\Omega$, $C_1 = 20pF$ and $L_1 = 20pH$. The implementation procedure in this example is similar to that of the example in Section 5.1.2, except for the part of calculating the voltage $VEi2$ which is the voltage returned from TLM_2 to the PSpice circuit. $VEi2$ is the convolution of VEr2 and G_{11} using Equation (5.11) instead of the direct connection between PSpice and TLM2.

$$VEi2 = VEr2 * G_{11} \tag{5.11}$$

where "$*$" means convolution.

Figure 5-16 shows that simulation results obtained with the convolution algorithm are in excellent agreement with those produced by MEFiSTo-2D.

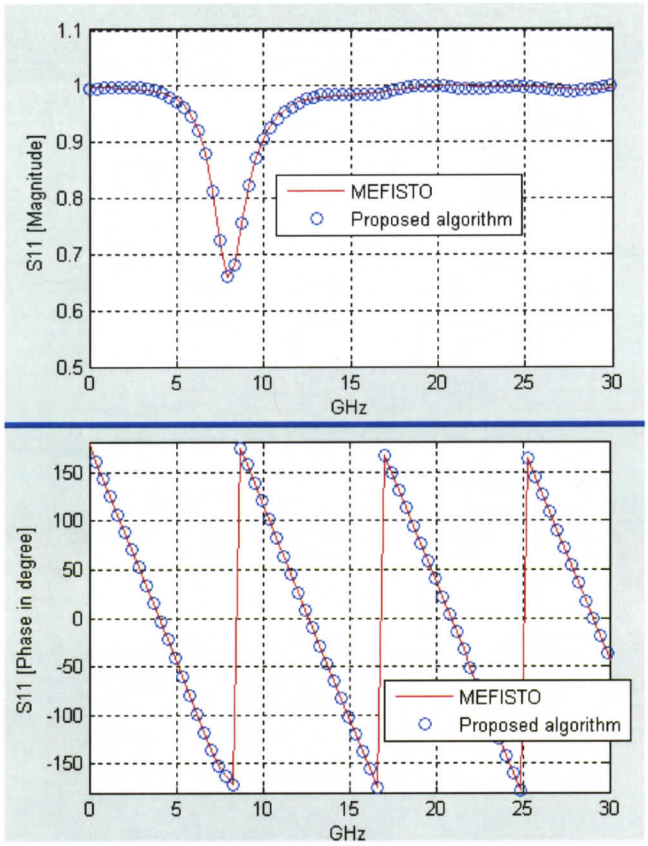


Figure 5-16 The simulation of the interconnection of TLM_1, TLM_2 and PSpice circuit by convolution labelled "Proposed algorithm" and results from MEFiSTo-2D "MEFISTO".

5.3 Direct Two-Port Connection between TLM and PSpice

In this section, a direct connection of two PSpice circuits ($PSpice_1$ (circuit1.cir) and $PSpice_2$ (circuit2.cir)) through a section of TEM waveguide TLM_2 is proposed. Another section of TEM waveguide is added to the front of the whole structure to calculate the reflection coefficient S_{11} as shown in Figure 5-17. TLM_2 is connected to $PSpice_1$ at interface1 and to $PSpice_2$ at interface2. The whole structure is modeled using the MEFiSTo-2D as shown in Figure 5-18. The two PSpice circuits, $PSpice_1$ and $PSpice_2$ are identical. Each circuit consists of a resistor, a capacitor and an inductor that are connected in shunt. TLM_1 and TLM_2 are two TEM waveguides of size $20mm \times 20mm$.

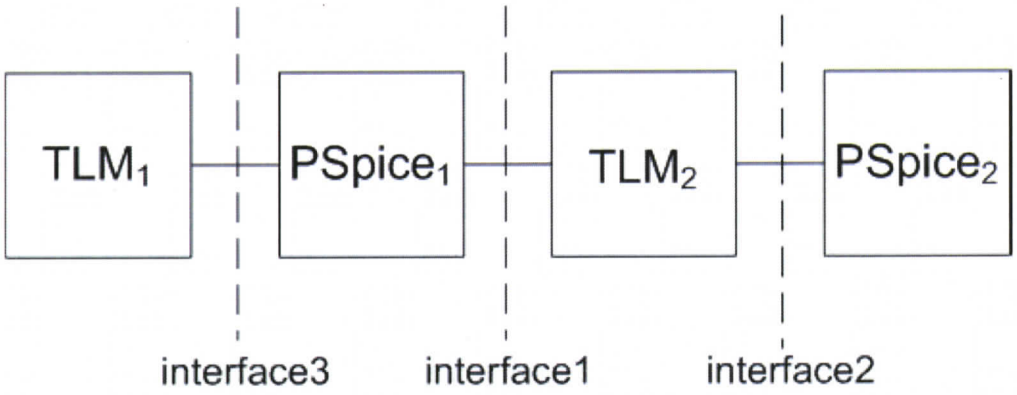


Figure 5-17 Direct connection of two TEM parallel plate waveguides TLM_1, TLM_2, and two PSpice circuits $PSpice_1$ and $PSpice_2$.

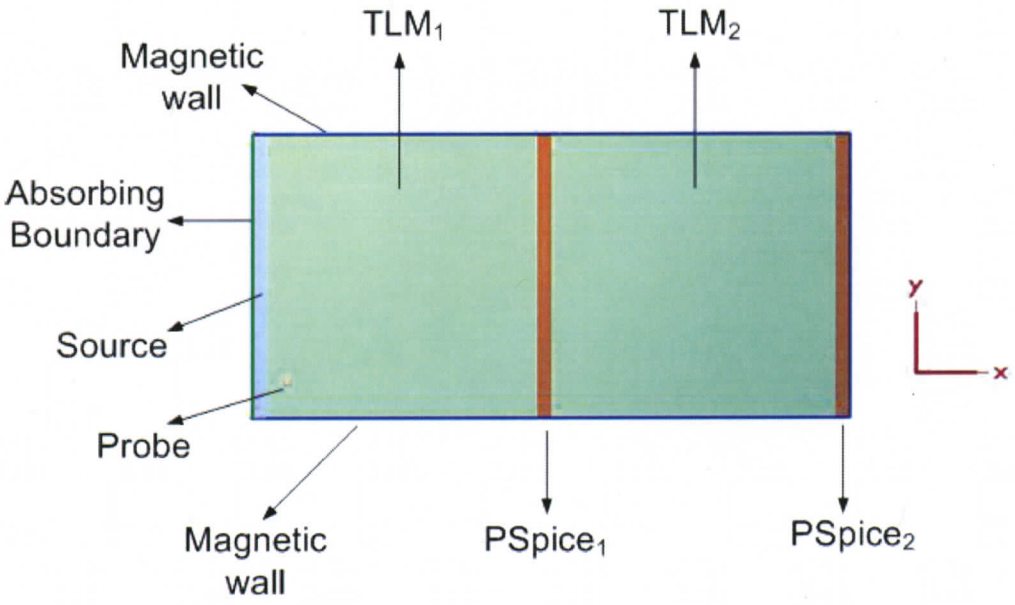


Figure 5-18 Direct connection of TLM_1, TLM_2, $PSpice_1$ and $PSpice_2$ modeled by MEFiSTo-2D.

To derive the equivalent circuit of the TLM and PSpice connection, special consideration is given to determine the Thévenin equivalent source of the TLM structure. Since the section of TEM waveguide TLM_2 is connected to both $PSpice_1$ and $PSpice_2$, one Thévenin equivalent source must be created for each port of TLM2. One port is connected to PSpice1, and the other to PSpice2.

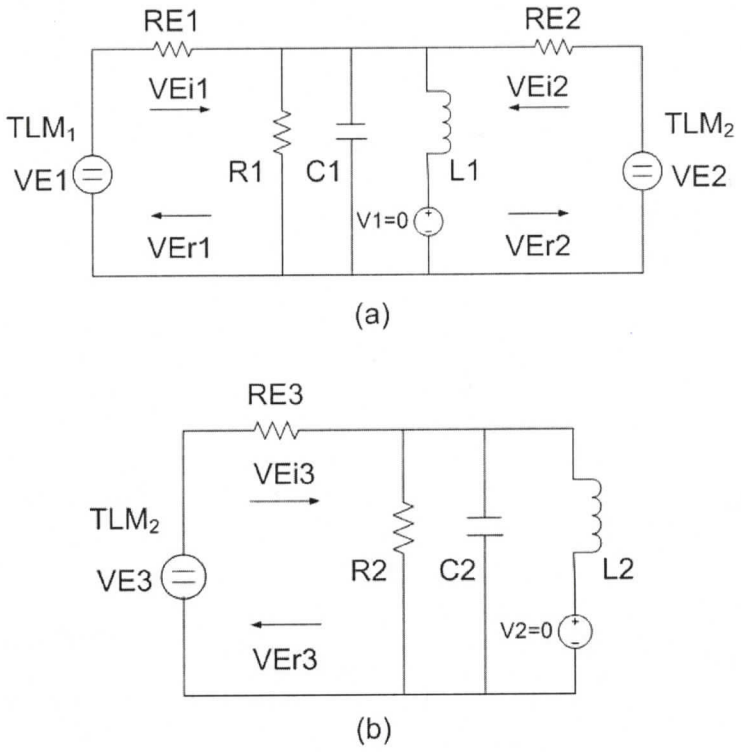


Figure 5-19 The equivalent circuit of the direct interconnection of TLM_1, TLM_2, $PSpice_1$ and $PSpice_2$ (a) TLM_1, $PSpice_1$ and left-half of TLM_2. (b) Right-half of TLM_2 and $PSpice_2$.

To analyze the whole structure, its equivalent circuit should be created first. The equivalent circuit of the whole structure is shown in Figure 5-19. Figure 5-19(a) represents the left part of the whole circuit (TLM_1, $PSpice_1$ and left-half of TLM_2); Figure 5-19(b) represents the right part of the whole circuit (right-half of TLM_2 and $PSpice_2$). $PSpice_1$ consists of resistor R_1, capacitor C_1 and inductor L_1. $PSpice_2$ consists of resistor R_2, capacitor C_2 and inductor L_2. The component values are $R_1 = R_2 = 5\Omega$, $C_1 = C_2 = 20pF$ and $L_1 = L_2 = 20pH$. Since the two TLM_2 half-subsections are identical, their two Thévenin equivalent impedances are the same, and therefore $RE2 = RE3$. RE1, RE2 and RE3 can be calculated using Equation (5.7), where n=20, since there are twenty removed branches at the TLM and PSpice interfaces, interface1 and interface2.

$$RE1 = RE2 = RE3 = \frac{Zl}{n} = \frac{Zo\sqrt{2}}{n} \approx \frac{377\Omega \times \sqrt{2}}{20} \approx 26.6579\Omega$$

All the voltages in the TLM mesh, VE1, VE2 and VE3 are set to zero at the beginning of the simulation. Circuit files circuit1.cir and circuit2.cir are created first. At time $t = k\Delta t$, VEi1 is the voltage emerging from TLM_1 at interface3; VEi2 is the voltage emerging from TLM_2 at interface1; VEi3 is the voltage emerging from TLM_2 at interface2. By Equation (5.1), the Thévenin equivalent voltage for TLM_1 (VE1) and the two subsections of TLM_2 (VE2 and VE3) are found by:

$$VE1 = 2 \times VEi1\,;\; VE2 = 2 \times VEi2\,;\; VE3 = 2 \times VEi3$$

PSpice is invoked (using the commands psp_cmd circuit1.cir and psp_cmd circuit2.cir) to simulate the equivalent circuit for one time-step. The resulting reflected voltages VEr1, VEr2 and VEr3 are returned to TLM network at time $t = (k+1)\Delta t$. The TLMs perform the scattering process for another time-step and determine the PSpice voltages VE1, VE2 and VE3 for the next time-step. PSpice is invoked to simulate the updated equivalent circuit and the cycle is repeated until the end of the simulation.

Figure 5-20 shows the S_{11} results: $|S_{11}|$ and $\angle S_{11}$ obtained with this direct connection and with MEFiSTo-2D. They are in excellent agreement.

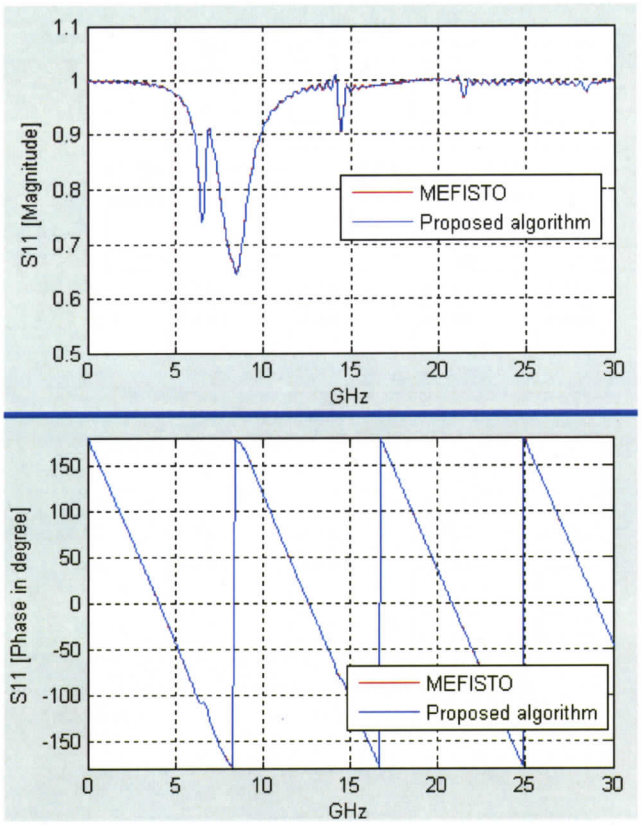


Figure 5-20 Simulation results obtained by direct connection of TLM_1, TLM_2, $PSpice_1$ and $PSpice_2$ labelled "Proposed algorithm" and results from MEFiSTo -2D labelled "MEFISTO".

5.4 Connection of Two-Port Johns Responses to PSpice by Convolution

In section 5.3, an example describing the connection of one-port Johns responses to PSpice by convolution was proposed. In this section, the connection of two-port Johns responses to PSpice by convolution is proposed. Two examples are given to describe the use of a two-port Johns Matrix when one TLM structure is connected to two PSpice circuits at two interfaces; and the use of a two-port Johns Matrix when one TLM structure is connected to a PSpice circuit and another TLM structure at two interfaces.

5.4.1 Test of Implementation 1

In the Section 5.3, the direct connection of two PSpice circuits, $PSpice_1$ and $PSpice_2$, and a section of TEM waveguide TLM_2 has been discussed. In this section, instead of using a direct connection, TLM_2 is characterized by a two-port Johns response first and then connected to the rest of the structure using convolution, as shown in Figure 5-21.

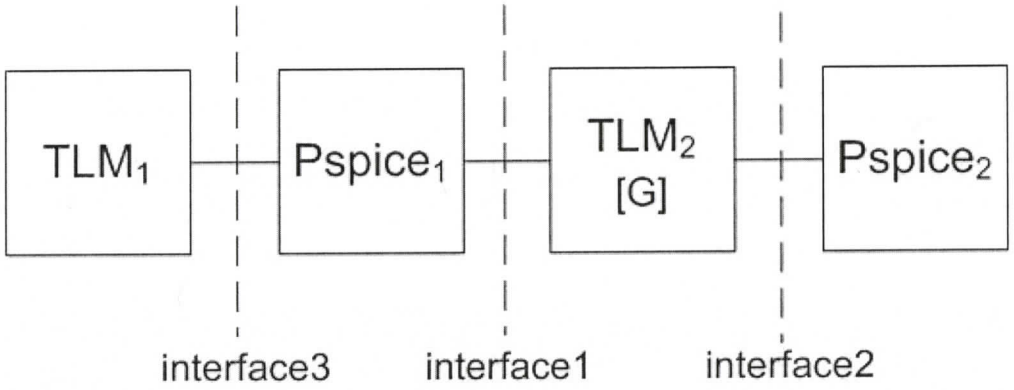


Figure 5-21 Connection of two TEM parallel plate waveguides TLM_1, TLM_2, and two PSpice circuits $PSpice_1$ and $PSpice_2$ by convolution.

The two-port Johns response of the TLM_2 is pre-calculated by analyzing TLM_2 in isolation. All voltages in the TLM_2 mesh are initially set to zero and unit impulses are injected into TLM_2 at interface1. The voltage impulses emerging from interface1 represent the reflected stream G_{11}; the voltage impulses emerging from interface2 represent the transmission stream G_{21}. After the Johns response of TLM_2 is found, the system shown in Figure 5-21 is converted to an equivalent circuit as shown in Figure 5-22. Since the section of TEM waveguide TLM_2 is connected to both $PSpice_1$ and $PSpice_2$, it is inconvenient to use one Thévenin equivalent source and Thévenin equivalent impedance to express TLM_2. To facilitate the analysis, TLM_2 is divided into two subsections. Figure 5-22(a) represents the equivalent circuit for TLM_1, $PSpice_1$ and the left-half of TLM_2 (stored in circuit1.cir). Figure 5-22(b) represents the right-half of TLM_2 and $PSpice_2$ (stored in circuit2.cir).

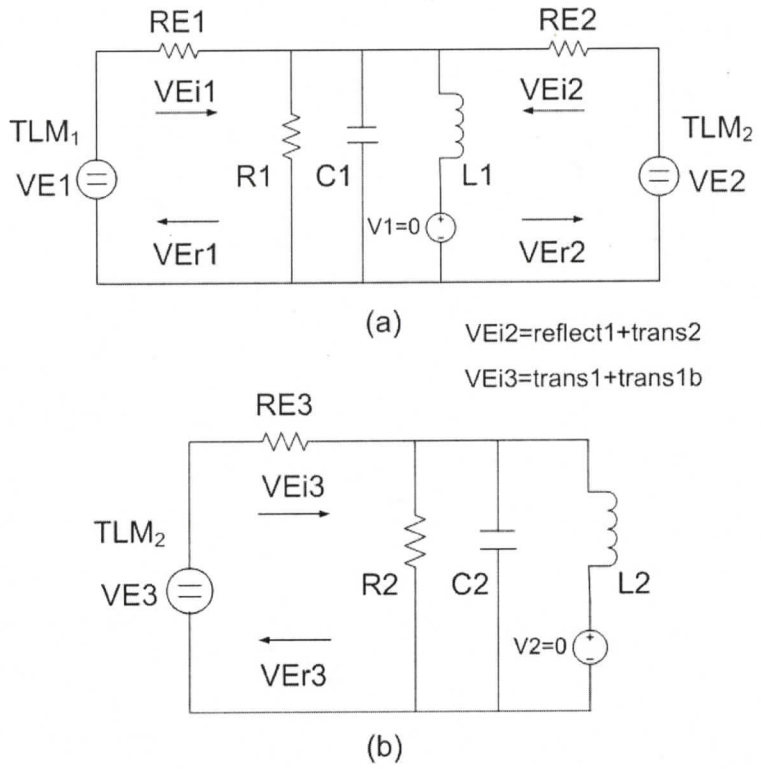


Figure 5-22 Equivalent circuit of the connection of TLM_1, TLM_2, $PSpice_1$ and $PSpice_2$ by convolution.

At the beginning of the simulation, VE1, VE2, VE3 and all the voltages in the mesh are set to zero. Since VEi1 is the voltage emerging from TLM_1 at the interface3 at time $t = k\Delta t$, the Thévenin equivalent source VE1 is therefore $2 \times VEi1$. The equivalent circuit shown in Figure 5-22 (a) is simulated in PSpice (using the command psp_cmd circuit1.cir) over one time-step to calculate VEr2. *reflect1* are the voltages reflected from TLM_2, and trans1 are the voltages going through TLM_2. Based on VEr2, *reflect1* and *trans1* are obtained as follows:

$$reflect1 = VEr2 * G11$$

$$trans1 = VEr2 * G21$$

$$VEi3 = trans1 + trans1b$$

$$VE3 = 2 * VEi3$$

After VE3 is found, the equivalent circuit shown in Figure 5-22(b) is simulated in PSpice (using the command psp_cmd circuit2.cir) over one time-step to calculate

VEr3. *trans1b* are the voltages reflected from TLM_2, and *trans2* are the voltages going through TLM_2. *trans1b* and trans2 are calculated from VEr3:

$$trans1b = VEr3 * G22$$

$$trans2 = VEr3 * G12$$

$$VEi2 = reflect1 + trans2$$

$$VE2 = 2 \times VEi2$$

Figure 5-23 shows that the simulation results obtained with MEFiSTo-2D and those calculated with the whole structure using convolution. They are in excellent agreement.

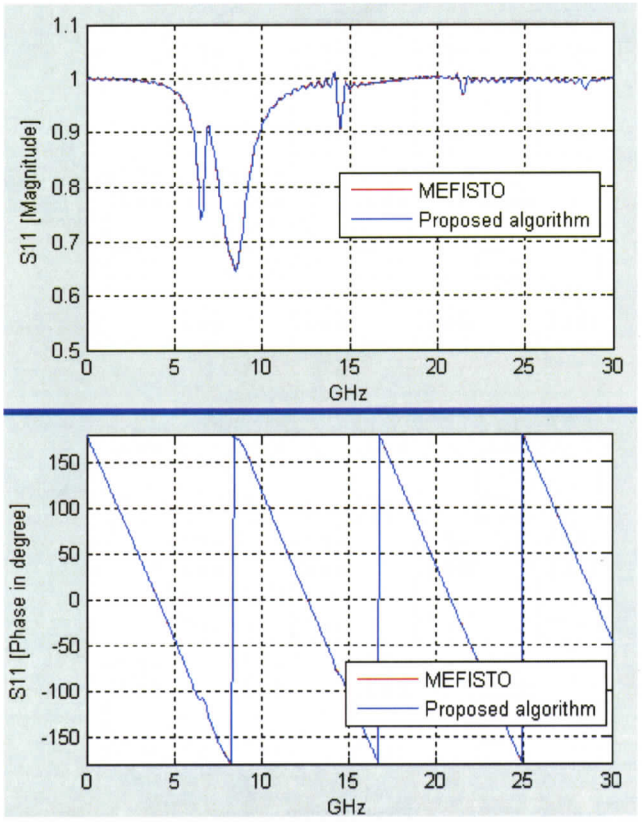


Figure 5-23 Simulation results of the connection of TLM_1, TLM_2, $PSpice_1$ and $PSpice_2$ by convolution labelled "Proposed algorithm" and results from MEFiSTo labelled "MEFISTO".

5.4.2 Test of Implementation 2

In this section, the connection of a two-port Johns response to PSpice by convolution is again considered using the example shown in Figure 5-24(a). A

section of TEM waveguide TLM_2 is connected to another section of TEM waveguide TLM_1 at interface1 and to one PSpice circuit at interface2. Both TLM_1 and TLM_2 have size $20mm \times 20mm$, with mesh size $1mm$ in both z- and x-directions. The PSpice circuit consists of a resistor R_1, a capacitor C_1 and an inductor L_1 connected in shunt, where $R_1 = 5\Omega$, $C_1 = 20pF$ and $L_1 = 20pH$.

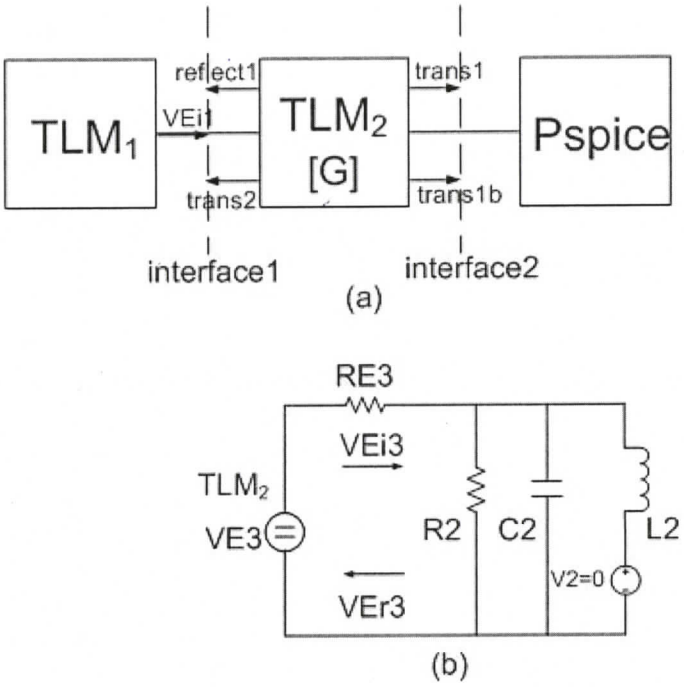


Figure 5-24 (a) An connection of two TEM parallel plate waveguides TLM_1 and TLM_2 with a PSpice circuit. (b) Equivalent circuit of the interconnection.

Figure 5-24(b) shows the equivalent circuit of the whole structure. All voltages in the TLM mesh are initialized to zero. VEi1 represent the voltage impulses emerging from TLM_1 at interface1. Based on VEi1, the voltages reflected from TLM_2 *reflect1*, and the voltages transmitted through TLM_2 *trans1* are derived as follows:

$$reflect1 = VEi1 * G_{11}$$

$$trans1 = VEi1 * G_{21}$$

$$VEi3 = trans1 + trans1b$$

After VEi3 is found, the Thévenin equivalent source voltage for TLM_2 VE3 can be found as:

$$VE3 = 2 * VEi3$$

Invoking the PSpice engine and letting PSpice calculate one time-step yields VEr3. *trans1b* are the voltages reflected from TLM_2; and *trans2* are the voltages going through TLM_2. Based on VEr3, *trans1b* and *trans2* are given by:

$$trans1b = VEr3 * G_{22}$$

$$trans2 = VEr3 * G_{12}$$

VEi2 represent the voltages going back to TLM_1 which can be found using:

$$VEi2 = reflect1 + trans2$$

Figure 5-25 shows that the S_{11} simulation results obtained by convolution are in great agreement with those obtained by MEFiSTo-2D.

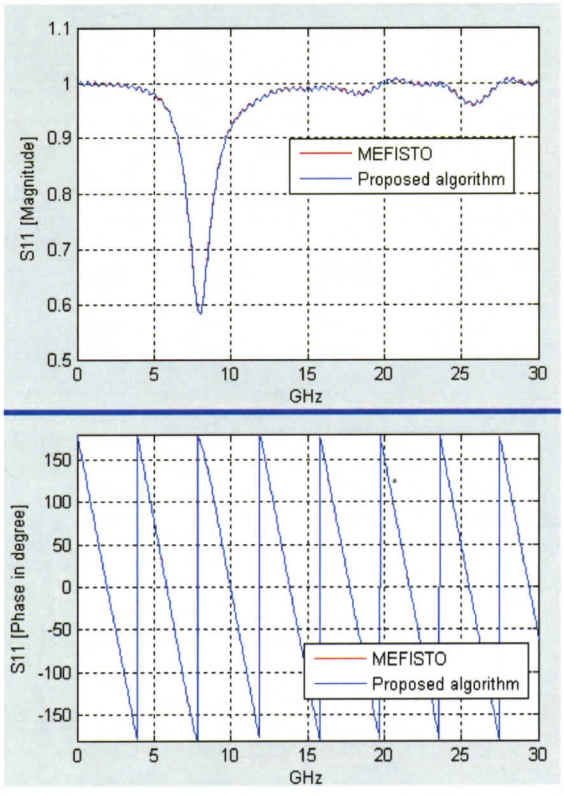


Figure 5-25 The simulation results of the interconnection of TLM_1, TLM_2 and PSpice by convolution labelled "Proposed algorithm" and results from MEFiSTo-2D labelled "MEFISTO".

Chapter 6

Validation of the Implementation

In Chapter 4 and Chapter 5, algorithms for connecting TLM structures by time domain diakoptics and for connecting TLM structures with PSpice circuits were proposed. In this chapter, two validation examples for the implementation of the algorithms will be presented. The first example is a single stub tuner in a TEM parallel plate waveguide; the second example is an inductive iris in a rectangular TE_{10} waveguide.

6.1 Example 1: Single Stub Tuning (TEM mode)

6.1.1 Theory

Single stub tuning is a matching technique that uses a single open-circuited or short-circuited transmission line to connect to another transmission line in shunt or in series. The length of the open-circuited stub or short-circuited stub and the distance from the location of the stub to the load can be adjusted to maximize power delivered to the load [42]. The advantage of the single shunt tuning stub is that it is easy to fabricate in microstrip or stripline since a via hole through the substrate to the ground plane is not needed.

A single short-circuited shunt stub will be discussed in this section. Figure 6.1 (a) shows a diagram of such a circuit; Figure 6.1(b) shows its 3D geometry. The load consists of a resistor and a capacitor connected in series, forming a load impedance $Z_L = 60 - j80\ \Omega$ at the operating frequency of 2GHz. To match this load to a 50Ω transmission line, a single short-circuited tuning stub is designed by first normalizing the load impedance:

$$z_L = \frac{Z_L}{50\Omega} = 1.2 - j1.6 .$$

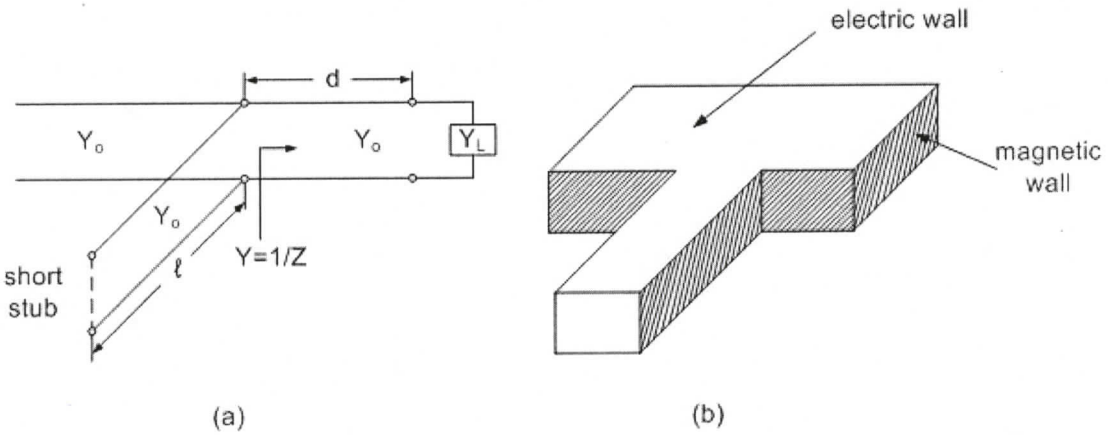


Figure 6-1 (a) Diagram of a single shunt short stub tuning circuit, (b) 3D structure of the single shunt short stub tuning circuit.

Since the stub is connected in parallel with the transmission line, it is convenient to calculate the admittance of the load, and plot it in the Smith Chart. The basic idea is to determine distance between the load and the stub, d, and stub length l, so that the admittance, Y (in the form of $Y_o + jB$), seen into the line at a distance d from the load, and susceptance of the stub result a matching condition. Therefore, the susceptance of the stub is chosen as $-jB$. The load is moved towards the generator until $Y = Y_o = 1$, since the normalized characteristic impedance of the transmission line is equal to 1. After distance d is determined, the stub length is chosen such that its input admittance is $-jB$ to make a matching condition. There are two sets of solutions to match the assumed load to a 50Ω line:

$d_1 = 0.11\lambda$, $l_1 = 0.095\lambda$, and

$d_2 = 0.26\lambda$, $l_2 = 0.405\lambda$

The two sets of solutions are shown in Figure 6.2.

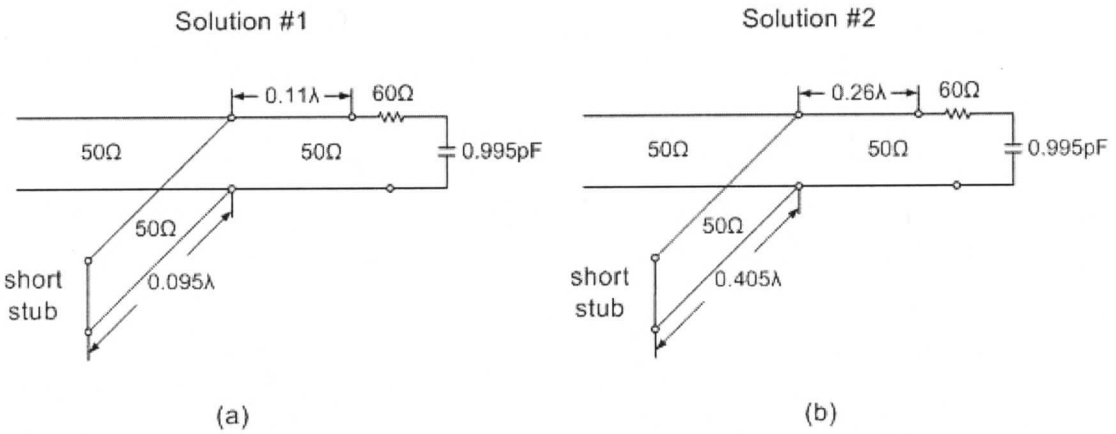


Figure 6-2 Two sets of solutions for the single stub tuning example.

6.1.2 Implementation and Results

Figure 6.3 shows the structure of a single short-circuited shunt stub and a reference structure modeled by MEFiSTo-2D using a 2D TLM shunt mesh. However, simulating the whole structure at once takes a longer time than dividing the whole structure into smaller subsections for individual analysis and reconnecting them together using convolution. In this example, the whole big structure is divided into three pieces: a small piece of TLM structure which contains the source, another piece of TLM structure that contains the stub, and a PSpice circuit that contains the load. The middle piece of the TLM structure is first characterized using a Johns matrix. It is then connected to the other TLM structure and the PSpice circuit using convolution.

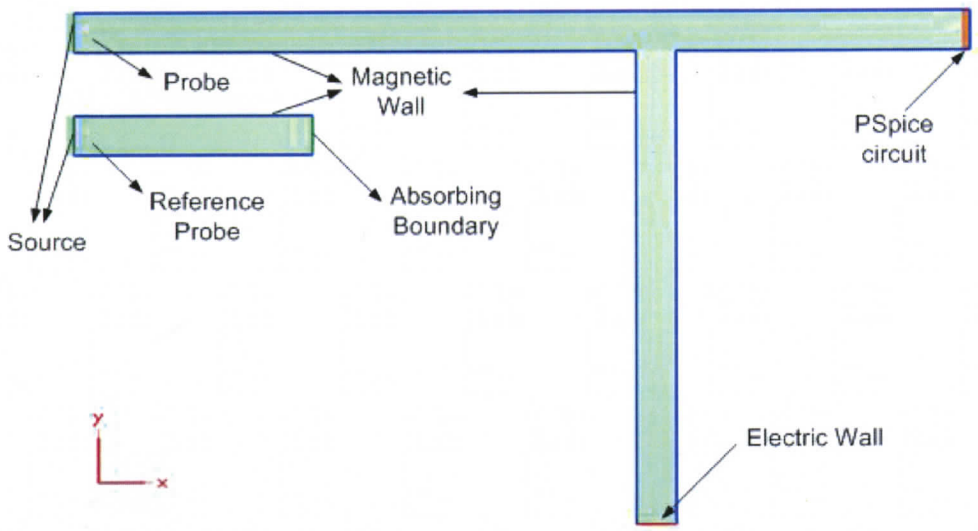


Figure 6-3 The structure of a single short-circuited shunt stub and a reference structure modeled by MEFiSTo-2D using a 2D TLM shunt mesh.

The Johns matrix of the middle piece of TLM structure is obtained by probing its input and output ports as shown in Figure 6.4. Assuming that the size of the input port is 5mm, and the mesh size is 1mm, there are five removed branches at the input port. Five probes are placed at the input port to extract the modal response. Since a combination of fundamental mode and higher evanescent order modes are present in the structure, modal extraction is applied to extract the fundamental mode in order to calculate the reflection Johns response G_{11} and the transmission Johns response G_{21} using Equation (4.3) in Chapter 4. After the Johns matrix of the middle structure has been obtained, the middle structure is connected to the other two structures using a convolution algorithm.

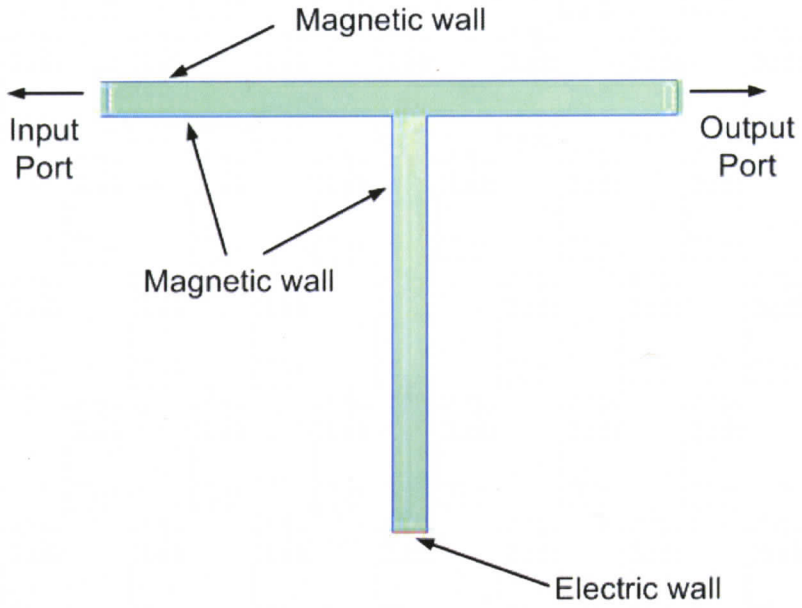


Figure 6-4 The Johns matrix of the middle structure for the single stub tuning example.

To develop the equivalent PSpice circuit, the link line impedance of the middle TLM structure is calculated by:

$$Z_l = Z_n * \sqrt{2} = 50\Omega * \sqrt{2} \tag{6.1}$$

where $Z_n = 50\Omega$ is the characteristic impedance of the transmission line. The height h of the structure is calculated using Equation (6.2).

$$Z_n = Z_o * \frac{h}{W}\sqrt{\frac{\mu_r}{\varepsilon_r}} \tag{6.2}$$

where Z_o is the electromagnetic wave impedance in free space, $Z_o \approx 377\Omega$; W is the width of the structure, $W = 5mm$; ε_r is relative permittivity; and μ_r is relative magnetic permeability. In air, $\varepsilon_r = \mu_r = 1$. So, the height of the structure is $h \approx 0.664mm$. The equivalent circuit of the whole structure is shown in Figure 6.5. The flow chart of the algorithm is shown in Figure 6.6.

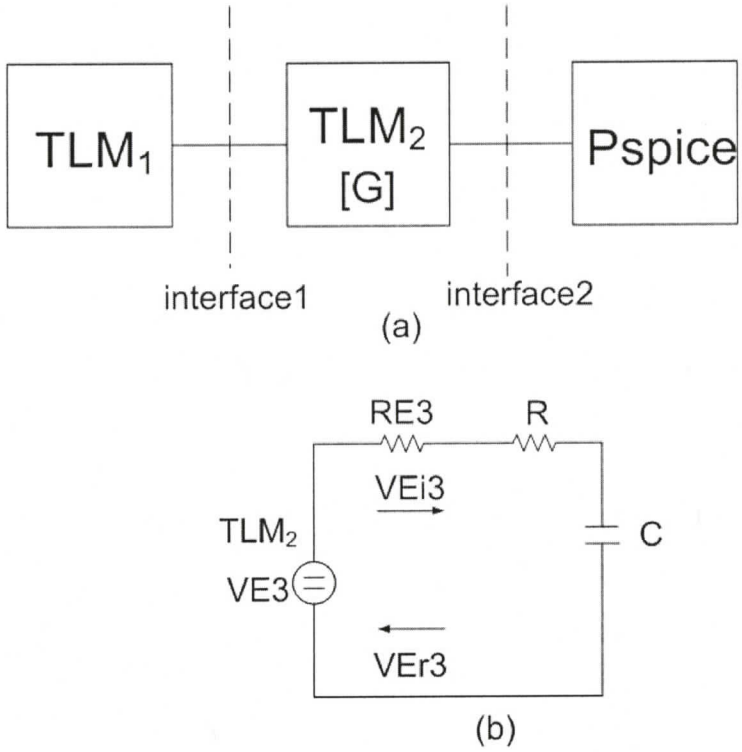


Figure 6-5 The equivalent circuit of the single stub tuner.

Prior to simulating the whole structure, the reflection Johns response G_{11} and the transmission Johns response G_{21} are pre-calculated as discussed above. PSpice circuit file is read and saved in a 2D array. The circuit file is updated by updating the corresponding element in the 2D array. For each time-step of the simulation, TLM_1 does initialization, scattering, connection and boundary condition to calculate voltages incident to the PSpice circuit VEi3. The Thévenin equivalent voltage source VE3 is based on VEi3. Initial condition (IC) of capacitor in the PSpice circuit is calculated by PSpice in the previous time-step. When PSpice circuit is updated, PSpice simulates the equivalent circuit for one time-step to calculate VEr3 which is the reflected voltage returned to TLM_1 during the next time-step. The voltages incident from TLM_1 to the PSpice circuit is calculated using convolution algorithm as described in the previous chapter, Section 5.4.2.

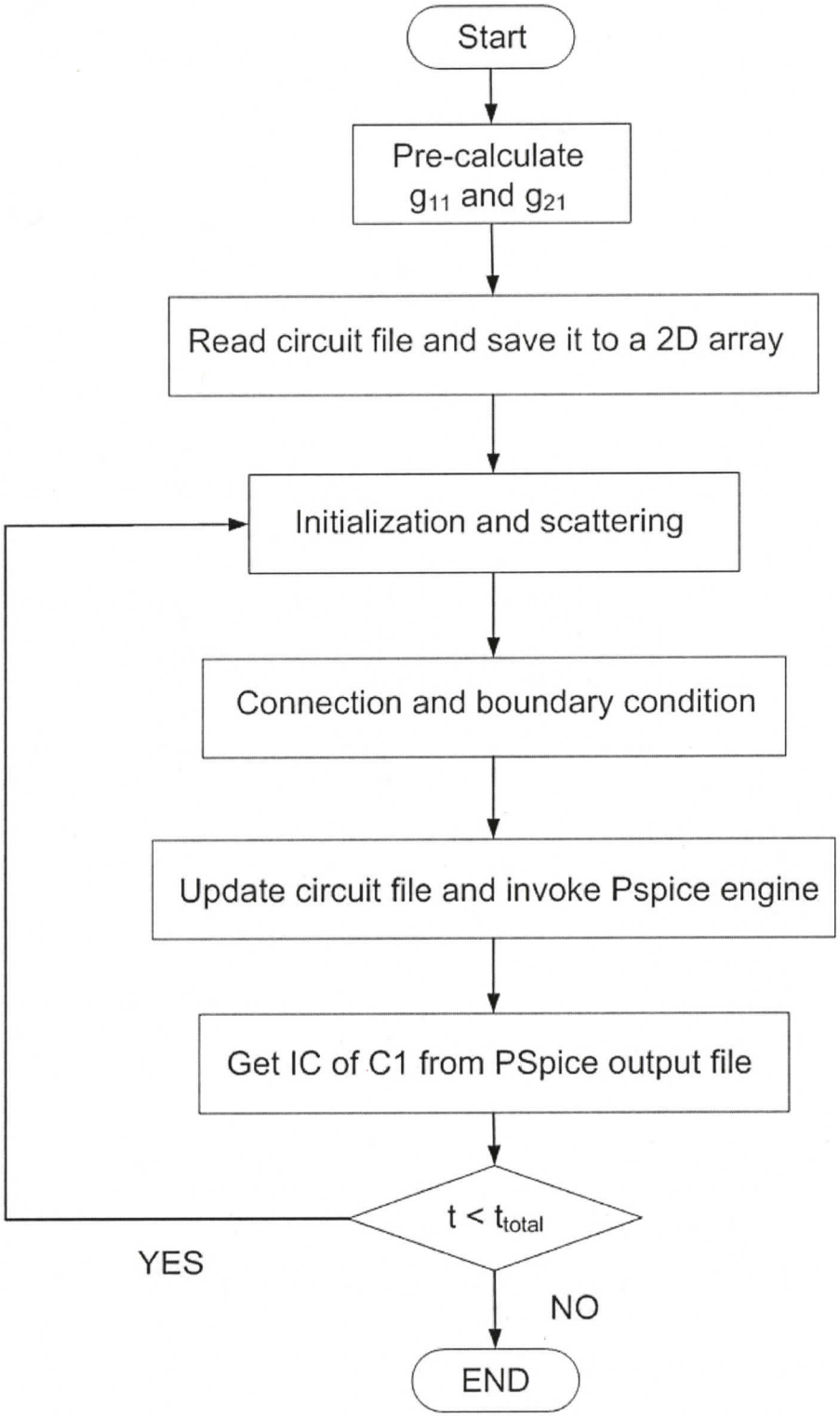


Figure 6-6 The flow chart of the diakoptics algorithm for the single stub tuning example.

Figure 6.7 shows that the S_{11} simulation results obtained with the convolution algorithm are in good agreement with the results from MEFiSTo-2D. The minimum of $|S_{11}|$ occurs at 2GHz, since the load is operating at a frequency of 2GHz. The slight discrepancy between the results is due to the fact that the MEFiSTo-2D simulation includes the effect of higher order evanescent modes generated at the T-junction.

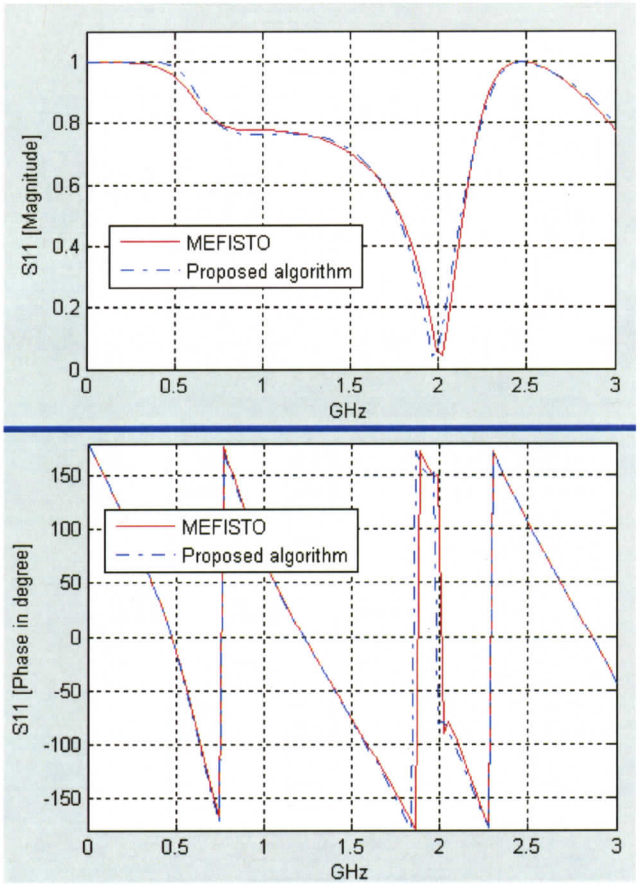


Figure 6-7 S_{11} results obtained with the convolution algorithm labelled "Proposed algorithm" and results from MEFiSTo labelled "MEFISTO".

6.2 Example 2: Inductive Iris in Rectangular TE_{10} waveguide

6.2.1 Theory

An inductive iris in the waveguide is formed by two thin metal plates placed in the middle of the waveguide. The two thin metal plates are perpendicular to the waveguide walls [11]. Figure 6.8 shows the geometry of the

discontinuity and the top view of the inductive iris in the rectangular waveguide modeled by MEFiSTo-2D using the 2D TLM shunt node.

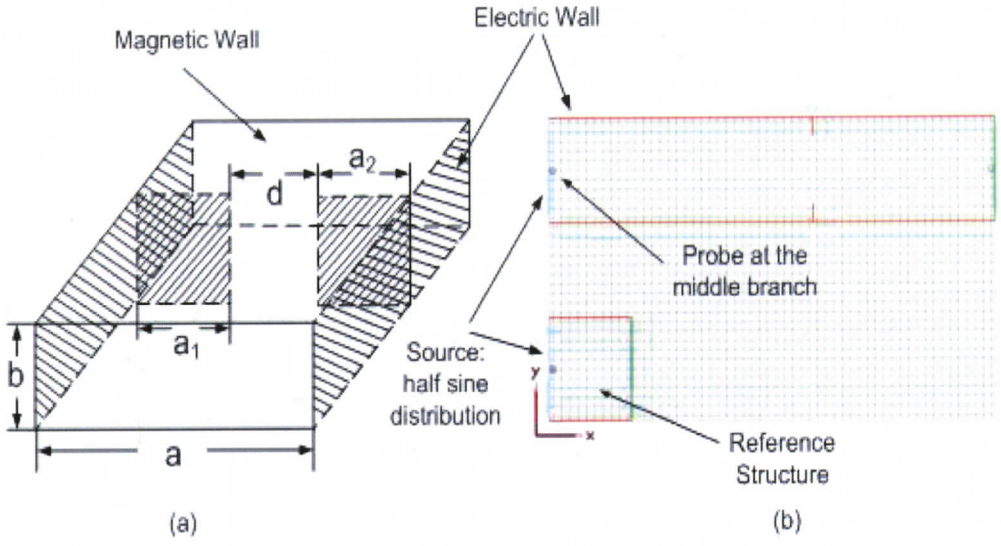


Figure 6-8 (a) Inductive iris in a rectangular waveguide. (b) Top view of the inductive iris in the rectangular waveguide as modeled in MEFiSTo-2D using 2D shunt TLM.

6.2.2 Implementation and Results

Direct simulation of the whole structure takes more time than the simulation of smaller substructures connected using convolution. The whole structure is divided into two smaller substructures TLM_1 and TLM_2 as shown in Figure 6-9. The Johns response of TLM_2 is pre-calculated based on probes placed at the input port as shown in Figure 6-10. After the Johns response of TLM_2 is found, TLM_1 and TLM_2 are reconnected together using convolution as shown in Figure 6.9. Since the inductive iris is present in the waveguide, not only the fundamental TE_{10} mode, but also higher order modes are excited by the iris in the waveguide. The fundamental TE_{10} mode is extracted using Equation (4.11) in Section 4.1.2 from the probe voltage at the input port.

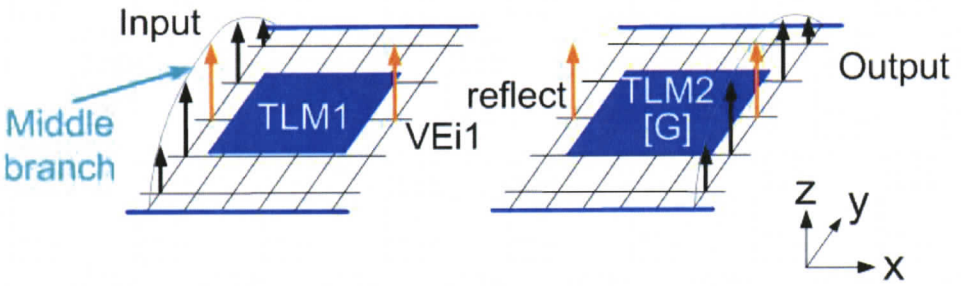


Figure 6-9 A diagram which illustrates the interconnection between TLM_1 and TLM_2.

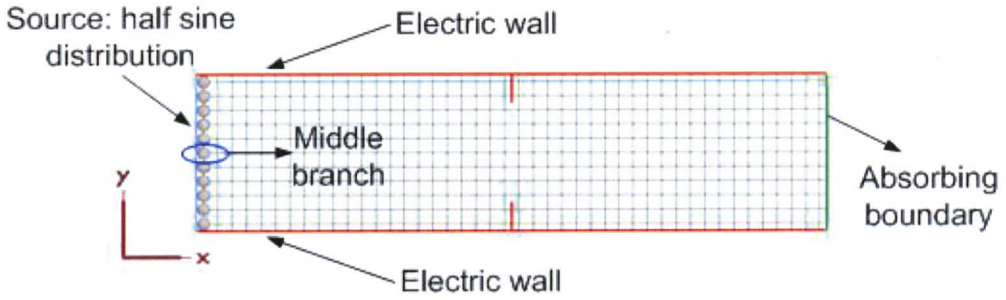


Figure 6-10 The Johns matrix of the middle structure for the inductive iris in the waveguide example.

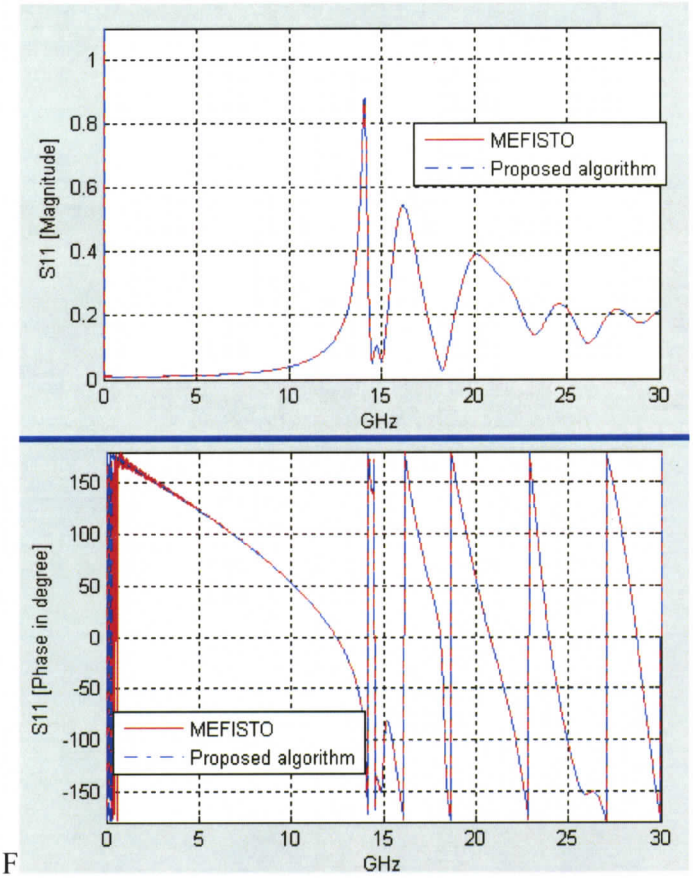
At the beginning of the simulation, all the voltages in the mesh are set to zero. A half-sine distribution is injected into TLM_1 at time t=0. *VEi1* is the voltage picked up at the middle branch in the TLM interface. The other voltages emerging from TLM_1 are calculated using the half-sine distribution according to the formula in Section 4.1.2:

$$\frac{A}{\sin\left(\frac{mid-0.5}{x_{\max}}\pi\right)}\sin\left(\frac{x-0.5}{x_{\max}}\pi\right)$$

where A is the amplitude of the voltage emerging from the middle branch; *mid* is the middle branch number; and x_{max} is the total branch number. After the Johns matrix of TLM_2 is found, voltages reflected from TLM_2 are calculated using convolution:

$$reflect = VEi1 * g_{11}$$

Figure 6.11 shows that the S_{11} results obtained with the convolution algorithm are in agreement with MEFiSTo-2D. The slight discrepancy between the results is due to the fact that the MEFiSTo-2D simulation accounts for higher-order modes present in the connecting interfaces.



F

Figure 6-11 S_{11} results obtained with the convolution algorithm labelled "Proposed algorithm" and results from MEFiSTo-2D labelled "MEFISTO".

Chapter 7

Conclusion and Future Work

In this chapter, the conclusion of the research work is presented. A summary of contributions and suggestions for future work is given.

7.1 Conclusions and Contributions

In this thesis, convolution algorithms for time domain diakoptics of TLM structures using one-port and two-port Johns Matrices for TEM and TE_{10} waveguides have been presented; algorithms for direct one-port and two-port connections between TLM and PSpice; and convolution algorithms for connecting TLM with PSpice have also been derived and validated.

For time domain diakoptics of TLM structures using one-port and two-port Johns Matrices, a TEM waveguide and a TE_{10} waveguide application have been presented. The results obtained with the modal convolution procedure are in excellent agreement with results obtained by simulating the whole structure without partitioning. Furthermore, the convolution algorithms take much less time. It has also been shown how to extract the dominant mode response for both the TEM and the TE_{10} case when evanescent higher order modes are also present in the ports. However, for perfect results, a multi-mode convolution procedure would be required, as proposed in [29].

Convolution algorithms for interconnecting TLM with PSpice have also been derived for both one-port and two-port connections. Five validation examples for the different cases were given to demonstrate the accuracy of the proposed algorithms. The results obtained in these five examples demonstrate that convolution algorithms and algorithms for direct connection between TLM and

PSpice yield results that are virtually identical with the results obtained for the whole structure with the electromagnetic simulator MEFiSTo-2D. The convolution algorithms take much less time. The objectives of this research work were thus accomplished successfully.

7.2 Future Work

The algorithms for the extraction of both TEM and TE_{10} mode responses are accurate enough (less than 1 percent) for most practical applications when evanescent higher-order modes are present in the connection interfaces. To improve the diakoptics procedure for such situations, higher-order mode convolution must be employed. The extension to multi-mode diakoptics would be one possible subject for future research.

The algorithms presented here are essentially based on 2D TLM structures, which assume that electric and magnetic fields have no variation in the third dimension. In more general cases, we need to extend the algorithms to 3D TLM structures. 3D TLM can model losses, inhomogeneous materials, and unequal link line impedances and is thus suitable for more generalized and more complex applications of our algorithms. The extension to 3D would require the use of two polarizations in TLM link connections and hence, would require two Johns responses instead of one. Furthermore, the extraction of modes from the field distribution of 3D ports would involve additional numerical effort. However, no fundamental changes are needed to extend the procedures to 3D.

Bibliography

[1] P. B. Johns, "A symmetrical condensed node for the TLM method," *IEEE Transactions on Microwave Theory and Techniques,* vol. 35, pp. 370-377, Apr 1987.

[2] W. J. R. Hoefer and P. P. M. So, *The MEFISTo-2D Theory*, 2nd ed. Victoria, British Columbia, Canada: Faustus Scientiffic Corporation, 1998.

[3] P. B. Johns and R. L. Beurle, "Numerical solution of 2-dimensional scattering problems using a transmission-line matrix," *Proceedings of the Institution of Electrical Engineers-London,* vol. 118, pp. 1203-&, 1971.

[4] G. Kron, *Diakoptics*, MacDonald, 1963.

[5] A. Braemellar, M. N. John and M. R. Scott, *Practical diakoptics for electrical networks*, Chapman and Hall, London, 1969.

[6] H. H. Happ, *Diakoptics and networks*, Academic Press, New York and London, 1971.

[7] S. Akhtarzad, "Analysis of lossy microwave structures and microstrip resonators by the TLM method," Ph.D dissertation, University of Nottingham, England, 1975.

[8] W. J. R. Hoefer, "The transmission line matrix method- theory and applications," *IEEE Transactions on Microwave Theory and Techniques,* vol. 33, pp. 882-893, 1985.

[9] P. B. Johns, "Solution of inhomogeneous waveguide problems using a transmission- line matrix," *IEEE Transactions on Microwave Theory and Techniques,* vol. TT22, pp. 209-215, 1974.

[10] H. Du, "Multilevel modeling techniques for time domain analysis of microwave and high speed circuits," Ph. D. Dissertation, University of Victoria, Victoria, 2006.

[11] C. Christopoulos, *The transmission-line modeling method TLM*, 1st ed. New York: IEEE Press, 1995.

[12] P. P. M. So, "Modeling of complex electromagnetic structures with TLM – theory and practice," Ph. D. Dissertation, University of Victoria, Victoria, British Columbia, Canada, 1996.

[13] P. P. M. So, "A new look at the 3D condensed node TLM scattering," *IEEE Intl. Microwave Symposium Digest,* pp. 1443-1446, 1993.

[14] G. Tardioli and W. J. R. Hoefer, "Derivation of the SCN node scattering matrix from the integral formulation of Maxwell's equations," *Proceeding of the Second International Workshop on Transmission Line Modeling (TLM)*, pp. 36-47, 1997.

[15] V. Trenkic, T. M. Benson and C. Christopoulos, "Dispersion analysis of a TLM mesh using a new scattering matrix formulation," *IEEE Microwave and Guided Wave Letters,* vol. 5, pp. 79-80, Mar 1995.

[16] V. Trenkic, C. Christopoulos and T. M. Benson, "Development of a general symmetrical condensed node for the TLM method," *IEEE Transactions on Microwave Theory and Techniques,* vol. 44, pp. 2129-2135, Dec 1996.

[17] M. Krumpholz and P. Russer, "A field-theoretical derivation of TLM," *IEEE Transactions on Microwave Theory and Techniques,* vol. 42, pp. 1660-1668, Sep 1994.

[18] V. Trenkic, C. Christopoulos and T. M. Benson, "Optimization of TLM schemes based on the general symmetrical condensed node," *IEEE Transactions on Antennas and Propagation,* vol. 45, pp. 457-465, Mar 1997.

[19] P. Naylor and R. Aitsadi, "Simple method for determing 3-D TLM nodal scattering in nonscalar problems," *Electronics Letters,* vol. 28, pp. 2353-2354, Dec 1992.

[20] V. Trenkic, C. Christopoulos and T. M. Benson, "Simple and elegant formulation of scattering in TLM nodes," *Electronics Letters,* vol. 29, pp. 1651-1652, Sep 1993.

[21] H. Jin and R. Vahldieck, "Direct derivations of TLM symmetrical condensed node and hybrid symmetrical condensed node from maxwells equations using centered differencing and averaging," *IEEE Transactions on Microwave Theory and Techniques,* vol. 42, pp. 2554-2561, Dec 1994.

[22] W. J. R. Hoefer, "The discrete time domain green's function or John's Matrix-a new powerful concept in transmission line modelling(TLM)," *International Journal of Numerical Modelling*, vol. 2, pp. 215-225, 1989.

[23] T. Okoshi, Y. Uehara and T. Takeuchi, "Segmentation method - approach to analysis of microwave planar circuits," *IEEE Transactions on Microwave Theory and Techniques,* vol. 24, pp. 662-668, 1976.

[24] R. Chadha and K. C. Gupta, "Segmentation method using impedance matrices for analysis of planar microwave circuits," *IEEE Transactions on Microwave Theory and Techniques,* vol. 29, pp. 71-74, 1981.

[25] R. Sorrentino, "Planar circuits, wave-guide models, and segmentation method," *IEEE Transactions on Microwave Theory and Techniques,* vol. 33, pp. 1057-1066, 1985.

[26] P. B. Johns and K. Akhtarzad, "The use of time domain diakoptics in time discrete models of fields," *International Journal for Numerical Methods in Engineering,* vol. 17, pp. 1-14, 1981.

[27] P. B. Johns and K. Akhtarzad, "Time domain approximations in the solution of fields by time domain diakoptics," *International Journal for Numerical Methods in Engineering,* vol. 18, pp. 1361-1373, 1982.

[28] Eswarappa, G. I. Costache and W. J. R. Hoefer, "Transmission-line matrix modeling of dispersive wide-band absorbing boundaries with time-domain diakoptics for S-parameter extraction," *IEEE Transactions on Microwave Theory and Techniques,* vol. 38, pp. 379-386, 1990.

[29] M. Righi and W. J. R. Hoefer, "Efficient 3D-SCN-TLM diakoptics for waveguide components," *IEEE Transactions on Microwave Theory and Techniques*, vol. 42, pp. 2381-2385, 1994.

[30] J. W. Park, P. P. M. So and W. J. R. Hoefer, "Lumped and distributed device embedding techniques in time domain," *IEEE MTT-S Int. Microwave Symp. Dig.*, 2001.

[31] S. J. Salon, M. J. Debortoli and R. Palma, "Couping of transient fields, circuits, and motion using finite-element analysis," *Journal of Electromagnetic Waves and Applications,* vol. 4, pp. 1077-1106, 1990.

[32] P. Zhou, W. N. Fu, D. Lin, S. Stanton and Z. J. Cendes, "Numerical modeling of magnetic devices," *IEEE Transactions on Magnetics,* vol. 40, pp. 1803-1809, 2004.

[33] H. Du, D. Gorcea, P. P. M. So and W. J. R. Hoefer, "A new analog behavioral module (ABM) linking field and PSpice-circuit simulations for transient analysis," *17th International Zurich Symposium on Electromagnetic Compatibility*, 2006.

[34] H. Du, P. P. M. So and W. J. R. Hoefer, "Embedding of current-coupled lumped networks in TLM models," *IEEE MTT-S Digest*, pp. 1705-1708, 2004.

[35] C. Christopoulos and J. L. Herring, "The application of Tranmission-Line Modeling (TLM) to electromagnetic compatibility problems," *IEEE*

Transactions on Electromagnetic Compatibility, vol. 35, pp. 185-191, 1993.

[36] P. Russer, M. Righi, C. Eswarappa and W. J. R. Hoefer, "Lumped element equivalent circuit parameter extraction of distributed microwave circuits via TLM simulation," *IEEE MTT-S Int. Microwave Symp. Dig.*, pp. 887-890, 1994.

[37] P. Russer, P. P. M. So and W. J. R. Hoefer, "Modeling of nonlinear active regions in TLM," *IEEE Microwave Guided Wave Lett.*, vol. 1, pp. 10-13, 1991.

[38] W. Eberle, P. Vandersteen, G. Wambacq, S. Donnay, G. Gielen and H. De Man, "Behavioral modeling and simulation of a mixed analog/digital automatic gain control loop in a 5GHz WLAN receiver," *Europe Conderence and Exhibition on Design, Automation and Test*, pp. 642-647, 2003.

[39] P. P. M. So and W. J. R. Hoefer, "A TLM-SPICE interconnection framework for coupled field and circuit analysis inthe time domain," *IEEE Transactions on Microwave Theory and Techniques*, pp. 2728-2733, 2002.

[40] P. P. M. So and W. J. R. Hoefer, "A generalized framework for SPICE-TLM interconnection," *IEEE MTT-S Int.Microwave Symp.*, 2002.

[41] J. M. Rabaey. (2001, May 10th). *SPICE*. http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/

[42] D. M. Pozar, *Microwave Engineering*, 3rd ed, New York: John Wiley & Sons Inc., 2005, Chapter 4 and 5.

Appendix A

Class Implementations

A.1 Class Array3D

```
////////////////////////////////////////////////////////////////////////////
//
// CLASS TEMPLATE: Array3D
//
// This is a generic three-dimensional array class.  Objects of this class
// can be treated as two- and one-dimensional arrays.  Array elements can be
// accessed using the () operator.  It is not necessary to use this class
// for one-dimensional array if the bulit-in C/C++ array meet your need.
//

#ifdef _Array3D_Examples_
// Examples
// --------
Array3D<int> a(10,10,10), b(a); // Create two 3D arrays of int, "a" and "b"
                                // have the same size (10x10x10) and
                                // dimension 3D
a = 0;                          // Assign "0" to all elements in "a"
b = a;                          // Copy values of "a" to "b"
b(3,4,5) = 7;                   // Assign a value to a specific entry in "b"

Array3D<double> c(10,10,1);     // Create a 2D array of double
c(2,1) = 4.0;                   // An example ssignment statement

Array3D<char> d(10,1,1);        // Create a 1D array of char
d = 'e';                        // Assign 'e' to all elments in "d": note
                                // that this is not possible if "d" is
                                // created using the built-in C/C++ syntax
                                // char d[10]
d(7) = 'k';                     // Change the 8th element to 'k'
#endif
//
////////////////////////////////////////////////////////////////////////////
#pragma once

#include <assert.h>

template<class T>
class Array3D
{
public:
        Array3D(const Array3D<T> &a)
        : i_size(a.i_size), j_size(a.j_size), k_size(a.k_size), b_size(a.b_size)
        {
```

```
            buffer = new T[b_size];
            *this = a;
        }
        Array3D(int I, int J, int K)
        : i_size(I), j_size(J), k_size(K), b_size(I*J*K)
        {
            buffer = new T[b_size];
        }
        ~Array3D()
        {
            delete[] buffer;
        }
        T& operator()(int i, int j, int k)const
        {
            assert(i>=0 && i<i_size && j>=0 && j<j_size && k>=0 && k<k_size);
            return buffer[(i*j_size+j)*k_size+k];
        }
        T& operator()(int i, int j)const
        {
            assert(i>=0 && i<i_size && j>=0 && j<j_size && k_size==1);
            return buffer[i*j_size+j];
        }
        T& operator()(int i)const
        {
            assert(i>=0 && i<i_size && j_size==1 && k_size==1);
            return buffer[i];
        }
        T& operator[](int i)const
        {
            assert(i>=0 && i<i_size && j_size==1 && k_size==1);
            return buffer[i];
        }

        Array3D<T>& operator=(const Array3D<T>& a)
        {
            assert (i_size==a.i_size && j_size==a.j_size &&
                            k_size==a.k_size &&     b_size==a.b_size);
            for (int n=0; n<b_size; n++) buffer[n]=a.buffer[n];
            return *this;
        }
        Array3D<T>& operator=(const T &v)
        {
            for (int n=0; n<b_size; n++) buffer[n]=v;
            return *this;
        }
        operator T*(){ return buffer; }
        operator const T*() const { return buffer; }

        const int i_size, j_size, k_size, b_size;

protected:
        T *buffer;
};
typedef Array3D<double> Double3D;
```

A.2 Class TLM Implementation

A.2.1 TLM.h

```
//////////////////////////////////////////////////////
//
//      Class TLM
//
//A TLM class handles electromagnetic field simulations in
//the fundamental TEM mode modeled by two-dimensional (2D)
//Transmission Line Matrix (TLM) method. A TLM class contains
//member functions for excitation, scattering, connection,
//boundary conditions, calculating Johns responses G11 and G12
//and calculating node voltages at a given node location.
//

#pragma once
#include "StdAfx.h"
#include "Array3D.h"
using namespace std;

class TLM
{
public:
        TLM(int, int, int); //Input: TLM mesh size and total simulation time
        void exitation(double); //Input: amplitude of voltage pulse
        void scattering(int); //Input: time step
        void connection();
        void boundary(double, double, double, double);//Input: boundary conditions

        //Output: transmit voltage impulses @ TLM mesh location (zmax-1,xmax/2)
        double transG12();

        //Output: reflect voltage impulse @ TLM mesh location (0,xmax/2);
        double reflectG11();

        //Calculate node voltage at a TLM mesh location
        void cal_vnode(int, int);
        ~TLM(void);

protected:
        int zmax, xmax, tmax, k; //(zmax, xmax): 2D TLM Mesh size; tmax: total
        //simulation time;k: number of timesteps
        double gama1, gama2, gama3, gama4;//Reflection coefficients of structure's
        //boundary
        double ex_amp; //Amplitude of field exitation
        double g11, g12; //g11: reflection voltage impulse at each time step
                         //g12: transmission voltage impulse at each time step

        Double3D Vi1, Vi2, Vi3, Vi4; //Incident voltage impulses to a node
        Double3D Vr1, Vr2, Vr3, Vr4; //Reflect voltage impulses from a node
        Double3D Vnode; //node voltage
        ofstream mynode;
};
```

A.2.2 TLM.cpp

```
#include "StdAfx.h"
#include "TLM.h"
using namespace std;

TLM::TLM(int zSize, int xSize, int totalTime) //mesh size and total simulation time
        : zmax(zSize), xmax(xSize), tmax(totalTime), Vi1(zSize, xSize,totalTime),
Vi2(Vi1), Vi3(Vi1), Vi4(Vi1), Vr1(Vi1), Vr2(Vi1), Vr3(Vi1), Vr4(Vi1),
Vnode(totalTime+3,1,1)
{
        Vi1 = 0.0;      Vi2 = 0.0;      Vi3 = 0.0;  Vi4 = 0.0;
        Vr1 = 0.0;  Vr2 = 0.0;  Vr3 = 0.0;  Vr4 = 0.0;
        g11=0.0, g12=0.0;

        ofstream file;
        file.open("node.txt",ios::out|ios::trunc); //Clear file node.txt
        file.close();
        mynode.open ("node.txt", ios::app); //Open file node.txt
}

//fundamental TEM mode: Inject unit impulses into input port of 2D TLM mesh
void TLM::exitation(double EX_AMP)
{
        ex_amp = EX_AMP;
        for (int x=0; x<xmax; x++)
        {
                Vi2(0,x,0)=ex_amp;
        }
}

void TLM::scattering(int TIME_STEP)
{
        k = TIME_STEP;
        for (int z = 0; z < zmax; z++)
        {
                for (int x = 0; x < xmax; x++)
                {
                        Vr1(z,x,k+1)=0.5*(-Vi1(z,x,k)+Vi2(z,x,k)+Vi3(z,x,k)+Vi4(z,x,k));
                        Vr2(z,x,k+1)=0.5*(Vi1(z,x,k)-Vi2(z,x,k)+Vi3(z,x,k)+Vi4(z,x,k));
                        Vr3(z,x,k+1)=0.5*(Vi1(z,x,k)+Vi2(z,x,k)-Vi3(z,x,k)+Vi4(z,x,k));
                        Vr4(z,x,k+1)=0.5*(Vi1(z,x,k)+Vi2(z,x,k)+Vi3(z,x,k)-Vi4(z,x,k));
                }
        }
}
void TLM::cal_vnode(int z_location, int x_location)
{
        Vnode[k]=0.5*(Vi1(z_location,x_location,k)+Vi2(z_location,x_location,k)+Vi3(z_lo
cation,x_location,k)+Vi4(z_location,x_location,k));
        mynode << k << "   "<< Vnode[k] << endl;
}
void TLM::connection()
{
        for (int z=0; z<zmax; z++)
        {
```

```
                for (int x=0; x<xmax; x++)
                {
                        if (x!=0)
                        {
                                Vi1(z, x, k+1)=Vr3(z, x-1, k+1);
                        }
                        if (z!=0)
                        {
                                Vi2(z, x, k+1)=Vr4(z-1, x, k+1);
                        }
                        if (x!=xmax-1)
                        {
                                Vi3(z, x, k+1)=Vr1(z, x+1, k+1);
                        }
                        if (z!=zmax-1)
                        {
                                Vi4(z, x, k+1)=Vr2(z+1, x, k+1);
                        }
                }
        }
}

void TLM::boundary(double GAMA1, double GAMA2, double GAMA3, double GAMA4)
{
        gama1=GAMA1; gama2=GAMA2; gama3=GAMA3; gama4=GAMA4;
        for (int z=0; z<zmax; z++)
        {
                for (int x=0; x<xmax; x++)
                {
                        if (gama1 == 0 || gama1 ==1 || gama1 ==-1 || gama1 ==-0.17157)
                        {
                                Vi1(z, 0, k+1)=gama1*Vr1(z, 0, k+1);
                        }
                        else Vi1(z, 0, k+1)=gama1;
                        if (gama2 == 0 || gama2 ==1 || gama2 ==-1 || gama2 ==-0.17157)
                        {
                                Vi2(0, x, k+1)=gama2*Vr2(0, x, k+1);
                        }
                        else Vi2(0, x, k+1)=gama2;
                        if (gama3 == 0 || gama3 ==1 || gama3 ==-1 || gama3 ==-0.17157)
                        {
                                Vi3(z, xmax-1, k+1)=gama3*Vr3(z, xmax-1, k+1);
                        }
                        else Vi3(z, xmax-1, k+1)=gama3;
                        if (gama4 == 0 || gama4 ==1 || gama4 ==-1 || gama4 ==-0.17157)
                        {
                                Vi4(zmax-1, x, k+1)=gama4*Vr4(zmax-1, x, k+1);
                        }
                        else Vi4(zmax-1, x, k+1)=gama4;

                }
        }
}
```

```
double TLM::transG12()
{
        g12=Vr4(zmax-1, xmax/2, k+1);
        return g12;
}

double TLM::reflectG11()
{
        g11=Vr2(0, xmax/2, k+1);
      . return g11;
}

TLM::~TLM(void)
{
        mynode.close();
}
```

A.3 Class TLM_TE Implementation

A.3.1 TLM_TE.h

```
//////////////////////////////////////////////////////////
//
//      Class TLM_TE
//
//A TLM_TE class is derived from class TLM and it handles
//electromagnetic field simulations in the fundamental
//TE10 mode modeled by 2D TLM method. It contains redefined
//functions for "exitation" and "boundary". A field profile
//with half-sine distribution are injected into the TLM
//mesh to excite TE10 mode.
//

#pragma once
#include "Array3D.h"
#include "TLM.h"
#include <math.h>
using namespace std;

class TLM_TE:public TLM
{
public:
        TLM_TE (int, int, int);
        void exitation(int); //Redefined "exitation" function for TE10 mode
        double cal_sum();
        //Redefined "boundary" function for TE10 mode
        void boundary(double, double, double, double);
      . void cal_vnode();
        Double3D cal_g11();

private:
        double pi;
        int number_branches;
};
```

A.3.2 TLM_TE.cpp

```
#include "StdAfx.h"
#include "TLM_TE.h"
using namespace std;

TLM_TE::TLM_TE (int xSize, int zSize, int totalTime) : TLM (xSize, zSize, totalTime)
{
        pi=3.1415926535897932384;
}

void TLM_TE::exitation(int EX_AMP)
{
        number_branches = EX_AMP;
        for (int x=0; x<xmax; x++)
        {
                Vi2(0,x,0)=sin(pi*(x+0.5)/number_branches);
        }
}
double TLM_TE::cal_sum()
{
        double sum=0;
        sum=Vr4(zmax-1,0,k+1)*sin(pi/22)+Vr4(zmax-1,1,k+1)*sin(3*pi/22)+Vr4(zmax-
1,2,k+1)*sin(5*pi/22)+Vr4(zmax-1,3,k+1)*sin(7*pi/22);
        sum=sum+Vr4(zmax-1,4,k+1)*sin(9*pi/22)+Vr4(zmax-1,5,k+1)*sin(11*pi/22)+Vr4(zmax-
1,6,k+1)*sin(13*pi/22)+Vr4(zmax-1,7,k+1)*sin(15*pi/22);
        sum=sum+Vr4(zmax-1,8,k+1)*sin(17*pi/22)+Vr4(zmax-
1,9,k+1)*sin(19*pi/22)+Vr4(zmax-1,10,k+1)*sin(21*pi/22);
        return sum;
}
void TLM_TE::boundary(double GAMA1, double GAMA2, double GAMA3, double GAMA4)
{
        gama1=GAMA1; gama2=GAMA2; gama3=GAMA3; gama4=GAMA4; //boundary conditions
        for (int z=0; z<zmax; z++)
        {
                for (int x=0; x<xmax; x++)
                {
                        if (gama1 == 0 || gama1 ==1 || gama1 ==-1 || gama1 ==-0.17157)
                        {
                                Vi1(z,0,k+1)=gama1*Vr1(z,0,k+1);
                        }
                        else Vi1(z,0,k+1)=gama1;
                        if (gama2 == 0 || gama2 ==1 || gama2 ==-1 || gama2 ==-0.17157)
                        {
                                Vi2(0,x,k+1)=gama2*Vr2(0,x,k+1);
                        }
                        else Vi2(0,x,k+1)=gama2;
                        if (gama3 == 0 || gama3 ==1 || gama3 ==-1 || gama3 ==-0.17157)
                        {
                                Vi3(z,xmax-1,k+1)=gama3*Vr3(z,xmax-1,k+1);
                        }
                        else Vi3(z,xmax-1,k+1)=gama3;
                        if (gama4 == 0 || gama4 ==1 || gama4 ==-1 || gama4 ==-0.17157)
                        {
                                Vi4(zmax-1,x,k+1)=gama4*Vr4(zmax-1,x,k+1);
```

```
                    }
                    else Vi4(zmax-1,x,k+1)=gama4*sin(pi*(x+0.5)/number_branches);

                }
        }
}

Double3D TLM_TE::cal_g11()
{
        Double3D g11_john(tmax+5,1,1);
        g11_john = 0;
        ifstream infile_pin1 ("pin1.txt");
        ifstream infile_pin2 ("pin2.txt");
        ifstream infile_pin3 ("pin3.txt");
        ifstream infile_pin4 ("pin4.txt");
        ifstream infile_pin5 ("pin5.txt");
        ifstream infile_pin6 ("pin6.txt");
        ifstream infile_pin7 ("pin7.txt");
        ifstream infile_pin8 ("pin8.txt");
        ifstream infile_pin9 ("pin9.txt");
        ifstream infile_pin10 ("pin10.txt");
        ifstream infile_pin11 ("pin11.txt");
        ofstream outfile_g11 ("g11.txt");
        string line_g11;
        double m1, m2, m3,sum_in;
        int index=0;
        if(infile_pin1&&infile_pin2&&infile_pin3&&infile_pin4&&infile_pin5&&infile_pin6&&
&infile_pin7&&infile_pin8&&infile_pin9&&infile_pin10&&infile_pin11)
        {
                for (int i=0; i<tmax; i++)
                {
                        sum_in=0;
                        getline (infile_pin1, line_g11);
                        stringstream (line_g11)>> m1 >> m2 >> m3;
                        sum_in=sum_in+m3*sin(pi/22);
                        getline (infile_pin2, line_g11);
                        stringstream (line_g11)>> m1 >> m2 >> m3;
                        sum_in=sum_in+m3*sin(3*pi/22);
                        getline (infile_pin3, line_g11);
                        stringstream (line_g11)>> m1 >> m2 >> m3;
                        sum_in=sum_in+m3*sin(5*pi/22);
                        getline (infile_pin4, line_g11);
                        stringstream (line_g11)>> m1 >> m2 >> m3;
                        sum_in=sum_in+m3*sin(7*pi/22);
                        getline (infile_pin5, line_g11);
                        stringstream (line_g11)>> m1 >> m2 >> m3;
                        sum_in=sum_in+m3*sin(9*pi/22);
                        getline (infile_pin6, line_g11);
                        stringstream (line_g11)>> m1 >> m2 >> m3;
                        sum_in=sum_in+m3*sin(11*pi/22);
                        getline (infile_pin7, line_g11);
                        stringstream (line_g11)>> m1 >> m2 >> m3;
                        sum_in=sum_in+m3*sin(13*pi/22);
                        getline (infile_pin8, line_g11);
                        stringstream (line_g11)>> m1 >> m2 >> m3;
```

```
                        sum_in=sum_in+m3*sin(15*pi/22);
                        getline (infile_pin9, line_g11);
                        stringstream (line_g11)>> m1 >> m2 >> m3;
                        sum_in=sum_in+m3*sin(17*pi/22);
                        getline (infile_pin10, line_g11);
                        stringstream (line_g11)>> m1 >> m2 >> m3;
                        sum_in=sum_in+m3*sin(19*pi/22);
                        getline (infile_pin11, line_g11);
                        stringstream (line_g11)>> m1 >> m2 >> m3;
                        sum_in=sum_in+m3*sin(21*pi/22);

                        if (index==0)
                        {
                                g11_john[index+1]=sum_in*2/11-1;
                        }
                        else g11_john[index+1]=sum_in*2/11;
                        outfile_g11 << index <<"   "<<g11_john[index]<< endl;
                        index=index+1;
                }
        }
        infile_pin1.close();
        infile_pin2.close();
        infile_pin3.close();
        infile_pin4.close();
        infile_pin5.close();
        infile_pin6.close();
        infile_pin7.close();
        infile_pin8.close();
        infile_pin9.close();
        infile_pin10.close();
        infile_pin11.close();
        outfile_g11.close();
        return g11_john;
}

void TLM_TE::cal_vnode()
{
        ///Calculate node voltage in TLM1
        Vnode[k]=0.5*(Vi1(0,(xmax-1)/2,k)+Vi2(0,(xmax-1)/2,k)+Vi3(0,(xmax-
1)/2,k)+Vi4(0,(xmax-1)/2,k));
        mynode << k << "   "<< Vnode[k] << endl;
}
```

A.4 Class Pspice_cir Implementation

A.4.1 Pspice_cir.h

```
////////////////////////////////////////////////////////
//
//      Class Pspice_cir
//
//Class Pspice_cir handles all the calculations involved
//with PSpice. It includes storing a PSpice netlist file to
//a 2D array, updating netlist file, involking PSpice engine
```

```
//and calculating the total voltage at the PSpice and TLM
//interface.
//

#pragma once
#include "Array3D.h"
using namespace std;

class Pspice_cir
{
public:
	//Input: two varibles to be updated and circuit file name
	Pspice_cir(string, string, string);

	//Input: three varibles to be updated and circuit file name
	Pspice_cir(string, string, string, string);

	//Input: four varibles to be updated and circuit file name
	Pspice_cir(string, string, string, string, string);

	//Input: Voltage impulse from one TLM structure to PSpice;
	//Update Thevenin equivalent source and initial conditions for capacitor and
	//inductor
	void update(string);

	//Input: Voltage impulse from two TLM structures to PSpice
	//Update Thevenin equivalent sources and initial conditions for capacitor and
	//inductor
	void update2(string, string);

	//Invoke PSpice engine and calculate total voltage at PSpice and TLM interface
	double calculate();

	//Invoke PSpice engine and calculate total voltage at PSpice and TLM interface
	double calculate2();

	~Pspice_cir(void);

	//The array netlist file is saved to
	Array3D<string> cir_array;

	double Vtotal; //Total voltage at the PSpice and TLM interface

private:
	//Line number of voltage sources, capacitors and inductors
	int counter_val1, counter_val2, counter_val3, counter_val4;

	//Line number of the line starting with VE, C or L
	int position1, position2, position3, position4;

	int counter; //Line number counter

	//Variables for netlist file
	string s1, s2, s3, s4, s5, s6;
```

```
        string line;

        //Local variables for voltage sources, capacitors and inductors
        string val1, val2, val3, val4, file_name;

        string cir2; //Name of temperary netlist file
        string C_IC1; //Initial conditions of capacitor
        string L_IC1; //Initial conditions of inductor
        string VE, VE2; //Inital conditions of voltage sources
};
```

A.4.2 Pspice_cir.cpp

```
#include "StdAfx.h"
#include "Pspice_cir.h"
using namespace std;

Pspice_cir::Pspice_cir(string string_val1,string string_val2,string
fileName):cir_array(25,6,1)
{
        val1 = string_val1;
        val2 = string_val2;
        file_name = fileName;
        string input_org = (file_name + ".cir").c_str();
        cout<<"file name:"<<input_org.c_str();
        ifstream infile (input_org.c_str());
        counter=0;
        counter_val1 = counter_val2 = counter_val3 = 0;
        if (infile)
        {
                while (!infile.eof())
                {
                        getline (infile,line);
                // Look for locations of string_val1 and string_val2 in the circuit file
                        position1 = int(line.find (string_val1,0));
                        position2 = int(line.find (string_val2,0));
                        s1=s2=s3=s4=s5=s6="";
                        if (position1 != string::npos)
                        {
                                stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
                                  cir_array(counter,0,0)=s1;
                                  cir_array(counter,1,0)=s2;
                                  cir_array(counter,2,0)=s3;
                                  cir_array(counter,3,0)=s4;
                                  cir_array(counter,4,0)=s5;
                                  cir_array(counter,5,0)=s6;
                                  counter_val1=counter;
                        }
                        else if (position2 != string::npos)
                        {
                                stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
                                  cir_array(counter,0,0)=s1;
                                  cir_array(counter,1,0)=s2;
                                  cir_array(counter,2,0)=s3;
```

```
                                cir_array(counter,3,0)=s4;
                                cir_array(counter,4,0)=s5;
                                cir_array(counter,5,0)=s6;
                                counter_val2=counter;
                        }
                        else
                        {
                             stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
                                cir_array(counter,0,0)=s1;
                                cir_array(counter,1,0)=s2;
                                cir_array(counter,2,0)=s3;
                                cir_array(counter,3,0)=s4;
                                cir_array(counter,4,0)=s5;
                                cir_array(counter,5,0)=s6;
                        }
                        counter=counter+1;
                }
             infile.close();
        }
    else cout << "Unable to open file";
        C_IC1 = "0";    L_IC1 = "0";    VE = "0";
}
Pspice_cir::Pspice_cir(string string_val1,string string_val2,string string_val3, string
fileName):cir_array(25,6,1)
{
        val1 = string_val1;
        val2 = string_val2;
        val3 = string_val3;
        file_name = fileName;

        string input_org = (file_name + ".cir").c_str();
        ifstream infile (input_org.c_str());
        counter=0;
        if (infile)
        {
                while (!infile.eof())
                {
                        getline (infile,line);
                //Look for the line starting with "VE3" in the circuit file input2.cir
                        position1 = int(line.find (string_val1,0));
                        position2 = int(line.find (string_val2,0));
                        position3 = int(line.find (string_val3,0));
                        s1=s2=s3=s4=s5=s6="";
                        if (position1 != string::npos)
                        {
                               stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
                                cir_array(counter,0,0)=s1;
                                cir_array(counter,1,0)=s2;
                                cir_array(counter,2,0)=s3;
                                cir_array(counter,3,0)=s4;
                                cir_array(counter,4,0)=s5;
                                cir_array(counter,5,0)=s6;
                                counter_val1=counter;
                        }
                        else if (position2 != string::npos)
```

```
                        {
                              stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
                                cir_array(counter,0,0)=s1;
                                cir_array(counter,1,0)=s2;
                                cir_array(counter,2,0)=s3;
                                cir_array(counter,3,0)=s4;
                                cir_array(counter,4,0)=s5;
                                cir_array(counter,5,0)=s6;
                                counter_val2=counter;
                        }
                        else if (position3 != string::npos)
                        {
                             stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
                                cir_array(counter,0,0)=s1;
                                cir_array(counter,1,0)=s2;
                                cir_array(counter,2,0)=s3;
                                cir_array(counter,3,0)=s4;
                                cir_array(counter,4,0)=s5;
                                cir_array(counter,5,0)=s6;
                                counter_val3=counter;
                        }
                        else
                        {
                             stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
                                cir_array(counter,0,0)=s1;
                                cir_array(counter,1,0)=s2;
                                cir_array(counter,2,0)=s3;
                                cir_array(counter,3,0)=s4;
                                cir_array(counter,4,0)=s5;
                                cir_array(counter,5,0)=s6;
                        }
                        counter=counter+1;
                }
            infile.close();
        }
    else cout << "Unable to open file";
        C_IC1 = "0";    L_IC1 = "0";    VE = "0";
}

Pspice_cir::Pspice_cir(string string_val1,string string_val2,string string_val3, string
string_val4, string fileName):cir_array(25,6,1)
{
        val1 = string_val1;
        val2 = string_val2;
        val3 = string_val3;
        val4 = string_val4;
        file_name = fileName;

        string input_org = (file_name + ".cir").c_str();
        ifstream infile (input_org.c_str());
        counter=0;
        if (infile)
        {
                while (!infile.eof())
                {
```

```
        getline (infile,line);
//Look for the line starting with "VE3" in the circuit file input2.cir
        position1 = int(line.find (string_val1,0));
        position2 = int(line.find (string_val2,0));
        position3 = int(line.find (string_val3,0));
        position4 = int(line.find (string_val4,0));
        s1=s2=s3=s4=s5=s6="";
        if (position1 != string::npos)
        {
            stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
              cir_array(counter,0,0)=s1;
              cir_array(counter,1,0)=s2;
              cir_array(counter,2,0)=s3;
              cir_array(counter,3,0)=s4;
              cir_array(counter,4,0)=s5;
              cir_array(counter,5,0)=s6;
              counter_val1=counter;
        }
        else if (position2 != string::npos)
        {
            stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
              cir_array(counter,0,0)=s1;
              cir_array(counter,1,0)=s2;
              cir_array(counter,2,0)=s3;
              cir_array(counter,3,0)=s4;
              cir_array(counter,4,0)=s5;
              cir_array(counter,5,0)=s6;
              counter_val2=counter;
        }
        else if (position3 != string::npos)
        {
            stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
              cir_array(counter,0,0)=s1;
              cir_array(counter,1,0)=s2;
              cir_array(counter,2,0)=s3;
              cir_array(counter,3,0)=s4;
              cir_array(counter,4,0)=s5;
              cir_array(counter,5,0)=s6;
              counter_val3=counter;
        }
        else if (position4 != string::npos)
        {
            stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
              cir_array(counter,0,0)=s1;
              cir_array(counter,1,0)=s2;
              cir_array(counter,2,0)=s3;
              cir_array(counter,3,0)=s4;
              cir_array(counter,4,0)=s5;
              cir_array(counter,5,0)=s6;
              counter_val4=counter;
        }
        else
        {
            stringstream (line)>> s1 >> s2 >> s3 >> s4 >> s5 >> s6;
              cir_array(counter,0,0)=s1;
```

```
                                cir_array(counter,1,0)=s2;
                                cir_array(counter,2,0)=s3;
                                cir_array(counter,3,0)=s4;
                                cir_array(counter,4,0)=s5;
                                cir_array(counter,5,0)=s6;
                        }
                        counter=counter+1;
                }
            infile.close();
        }
    else cout << "Unable to open file";
        C_IC1 = "0";    L_IC1 = "0";    VE = "0";       VE2 = "0";
}

void Pspice_cir::update(string VE)
{
        if (counter_val3 ==0)
        {
                cir_array(counter_val1,4,0)= VE;
                cir_array(counter_val2,4,0)="IC="+C_IC1;
        }
        else
        {
                cir_array(counter_val1,4,0)= VE;
                cir_array(counter_val2,4,0)="IC="+C_IC1;
                cir_array(counter_val3,4,0)="IC="+L_IC1;
        }

        string s = ("_"+file_name).c_str();
        cir2 = s + ".cir";
        ofstream outfile (cir2.c_str());
        for (counter=0; counter<25; counter++)
        {
                outfile << cir_array(counter,0,0)<<"    "<<cir_array(counter,1,0)<<"
"<<cir_array(counter,2,0)<<"  "<<cir_array(counter,3,0)<<"  "<<cir_array(counter,4,0)<<"
"<<cir_array(counter,5,0)<<endl;
        }
        outfile.close();
}

void Pspice_cir::update2(string VE, string VE2)
{
                cir_array(counter_val1,4,0)= VE;
                cir_array(counter_val2,4,0)= VE2;
                cir_array(counter_val3,4,0)="IC="+C_IC1;
                cir_array(counter_val4,4,0)="IC="+L_IC1;
                string s = ("_"+file_name).c_str();
                cir2 = s + ".cir";
                ofstream outfile (cir2.c_str());
                for (counter=0; counter<25; counter++)
                {
                        outfile << cir_array(counter,0,0)<<"
"<<cir_array(counter,1,0)<<"  "<<cir_array(counter,2,0)<<"  "<<cir_array(counter,3,0)<<"
"<<cir_array(counter,4,0)<<"  "<<cir_array(counter,5,0)<<endl;
                }
```

```
                outfile.close();
}

double Pspice_cir::calculate()
{
                string lineb;
                double tb, v1b, v2b, I;
                double ttb, vv1b, vv2b, II;
                int posb;

                system (("..\\PSpice\\psp_cmd "+cir2).c_str());//invoke PSPICE engine
                string file_out = ("_"+file_name).c_str();
                ifstream in_cir2 ((file_out + ".out").c_str());
                if (in_cir2)
                {
                        do
                        {
                                getline (in_cir2, lineb);
                                posb = int(lineb.find("0.000E+00", 0));
                        }while (!in_cir2.eof()&& posb== string::npos);
                        stringstream (lineb) >> tb >> v1b >> v2b >> I;
                       getline (in_cir2, lineb); //get the line where one time step ends
                        stringstream (lineb) >> ttb >> vv1b >> vv2b >> II ;
                        Vtotal = v2b+vv2b;
                        stringstream out2;
                        out2 << vv2b;
                        //Get the initial condition of capacitor for the next timestep
                        C_IC1 = out2.str();
                        stringstream out3;
                        out3 << II;
                        L_IC1 = out3.str();
                }
                else cout << "Unable to open file";
        return Vtotal;
}

double Pspice_cir::calculate2()
{
                string lineb;
                double tb, v1b, v2b;
                double ttb, vv1b, vv2b;
                int posb;

                system (("..\\PSpice\\psp_cmd "+cir2).c_str());//invoke PSPICE engine
                string file_out = ("_"+file_name).c_str();
                ifstream in_cir2 ((file_out + ".out").c_str());
                if (in_cir2)
                {
                        do
                        {
                                getline (in_cir2, lineb);
                                posb = int(lineb.find("0.000E+00", 0));
                        }while (!in_cir2.eof()&& posb== string::npos);
                        stringstream (lineb) >> tb >> v1b >> v2b;
                       getline (in_cir2, lineb); //get the line where one time step ends
```

```
                    stringstream (lineb) >> ttb >> vv1b >> vv2b ;
                    Vtotal = v1b+vv1b;
                    stringstream out2;
                    out2 << vv2b;
                    C_IC1 = out2.str();
                    //Get the initial condition of capacitor for the next timestep
                }
                else cout << "Unable to open file";
        return Vtotal;
}

Pspice_cir::~Pspice_cir(void)
{
}
```

A.5 Class TLM_Pspice Implementation

A.5.1 TLM_Pspice.h

```
////////////////////////////////////////////////////////
//
//      Class TLM_Pspice
//
//Class TLM_Pspice is for direct connection of TLM engine
//and PSpice circuits. It handles data exchanges between TLM
//engine and PSpice circuits.
//

#pragma once
using namespace std;

class TLM_Pspice
{
public:
        TLM_Pspice(void);

        //Input: Voltage impulse from TLM to PSpice;
        //Output: Thevenin equivalent source for TLM structure
        string toPspice(double,double);

        //Input: Total voltage at TLM and PSpice interface calculated by PSpice
        //Output: Voltage impulses from PSpice to TLM
        double toTLM(double);

        ~TLM_Pspice(void);
        string Vtlm; //Thevenin Equivalent source
private:
        double voltage_tlm; //voltage from TLM structure to PSpice
        double reflect_voltage; //voltage returned from PSpice to TLM
        double height_TLM; //Height of 2D TLM structure
};
```

A.5.2 TLM_Pspice.cpp

```
#include "StdAfx.h"
#include "TLM_Pspice.h"
#include <math.h>
using namespace std;

TLM_Pspice::TLM_Pspice(void)
{
        voltage_tlm = 0;
        reflect_voltage = 0;
        height_TLM = 0;
}
string TLM_Pspice::toPspice(double voltageTLM, double height)
{
        height_TLM = height;
        voltage_tlm = voltageTLM;
        stringstream out;
        out << voltage_tlm * 2 * height_TLM;
        Vtlm = out.str(); //convert double to string
        return Vtlm;
}

double TLM_Pspice::toTLM(double Vtotal)
{
        reflect_voltage = 0.5 * Vtotal/height_TLM - voltage_tlm;
        return reflect_voltage;
}

TLM_Pspice::~TLM_Pspice(void)
{
}
```

A.6 Class Conv_Pspice Implementation

A.6.1 Conv_Pspice.h

```
////////////////////////////////////////////////////////
//
//      Class Conv_Pspice
//
//Class Conv_Pspice is for the connection of TLM engine
//and PSpice circuits by convolution. It handles data
//exchanges between TLM engine and PSpice circuits.
//

#pragma once
#include "StdAfx.h"
#include "Array3D.h"
#include "TLM_Pspice.h"
using namespace std;

class Conv_Pspice: public TLM_Pspice
{
```

```
public:
        //Input: one number, an array, size of output and time step
        //Output: Convolution result
        Double3D Convolve(const double &, const Double3D &, int, int);

        //Input: two arrays to be convolved and size of output
        //Output: Convolution result
        Double3D Convolve_array(const Double3D &, const Double3D &, int);
};
```

A.6.2 Conv_Pspice.cpp

```
#include "StdAfx.h"
#include "Conv_Pspice.h"
using namespace std;

Double3D Conv_Pspice::Convolve(const double &a, const Double3D &b, int _size_, int k)
{
        assert(b.b_size>=_size_);
        Double3D c(_size_,1,1);
        c = 0.0;
        for (int j=0; j<_size_-k; j++)
        {
                c[j+k]=a * b[j] + c[j+k];
        }
        return c;
}
Double3D Conv_Pspice::Convolve_array(const Double3D &a, const Double3D &b, int _size_)
{
        assert(a.b_size>=_size_ && b.b_size>=_size_);
        Double3D c(_size_,1,1);
        c = 0.0;

        for (int i=0; i<_size_; i++)
        {
                for (int j=0; j<_size_-i; j++)
                {
                        c[j+i]=a[i] * b[j] + c[j+i];
                }
        }
        return c;
}
```

Appdendix B

Implementation Examples

B.1 Implementations for Section 4.1

```
//*********      Readme      ***********//
/*Purpose of the code:
      Combining two TEM structures TLM1 and TLM2 with the same size
      (zmax, xmax) and total simulation time is t1. Reflection coefficients of TLM
      structure's boundary are the input of boundary function  */
/*Goal: Prove the results from simulating the whole structure are the same as
          connection of TLM1 and TLM2 by convolution
Procedure: 1. Calculate Johns response G11 and G12 of one TLM structure TLM1
           2. Connect two TEM waveguides TLM1 and TLM2 by convolution
           3. Save the output voltage result in the text file output.txt
           4. Create the big structure TLM in MEFiSTo-2D to obtain the output
              voltage result
           5. Compare results from the proposed algorithm with MEFiSTo-2D.
*/

#include "Stdafx.h"
#include "Array3D.h"
#include "TLM.h"
#include "TLM_Pspice.h"
#include "Conv_Pspice.h"
using namespace std;

int _tmain(int argc, _TCHAR* argv())
{
        clock_t start = clock();
        int t1=1000;
        int zmax = 20, xmax = 20, tmax=t1+1;//2D TLM mesh size and total simulation time
        Double3D g11(t1+3,1,1), g12(g11), myout(g11);
        Double3D reflect(g11);
        g11=0.0; g12=0.0; myout=0.0; reflect=0.0; //Variable initialization

        // Calculate G11 and G12
        TLM TLM1(zmax, xmax, tmax);
        TLM1.exitation(1.0);
        for (int k = 0; k < t1; k++)
        {
                TLM1.scattering(k);
                g12[k+1] = TLM1.transG12();
                g11[k+1] = TLM1.reflectG11();
                TLM1.connection();
                TLM1.boundary(1,0,1,-0.17157); //reflection coefficients of boundary
        }
```

```
    // Three elementary TLM processes:
    //scattering, connection and reflection at boundary
        Conv_Pspice conv;
        TLM TLM2(zmax, xmax, tmax);
        TLM2.exitation(1.0);
        for (int k = 0; k < t1; k++)
        {
                TLM2.scattering(k);
                myout[k+1] = TLM2.transG12();
                for (int j=1; j<t1-k; j++)
                {
                        reflect[j+k]=myout[k+1]*g11[j]+reflect[j+k];
                }
                TLM2.connection();
                TLM2.boundary(1,-0.17157,1,reflect[k]);
        }

        // Save output to file output.txt
        ofstream myresult ("output.txt");
        Double3D result(t1,1,1);

        result = conv.Convolve_array(myout,g12,t1);
        for (int i=0; i<t1; i++)
        {
                myresult << result[i] << endl;
        }

        clock_t finish = clock();
        cout << endl << "Elapsed Time [sec] : " << (finish-start)/CLOCKS_PER_SEC;
        cout << endl << "Job done, hit <enter> to terminate the program.";
        cin.get();
        return 0;
}
```

B.2 Implementations for Section 5.1.2

```
//*********      Readme      ***********//
/*Purpose of the code:
          Direct one-port connection between TLM structure with size (zmax,xmax)
          and PSpice circuit which contains shunt connected resistor R1, capacitor C1
          and inductor L1
Goal:     Prove direct one-port connection between TLM structure and PSpice
          obtained from the proposed algorithm has a good agreement with MEFiSTo-2D
Procedure: 1. Create PSpice circuit file input.cir including shunt connected resistor,
              capacitor and inductor
           2. circuit1 is object of class Pspice_cir; TLM1 is object of class TLM;
              interface1 is object of class TLM_Pspice
           3. Node voltage at a specific node location in the TLM1 mesh (2,2)
              is saved to a text file node.txt
           4. Reference node voltage is saved to a text file ref.txt
           5. Create the same structure in MEFiSTo-2D and save node voltage at the same
              location and reference node voltage node_mef.txt and ref_mef.txt,
              respectively.
           6. Use matlab file myfft.m to calculate  S11  and S11 phase at the above mesh
```

```
            location (2,2) using the proposed algorithm and MEFiSTo-2D
          7. Compare results from the proposed algorithm and MEFiSTo-2D.
*/

#include "StdAfx.h"
#include "Array3D.h"
#include "TLM.h"
#include "Pspice_cir.h"
#include "TLM_Pspice.h"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
        clock_t start = clock();
        int t1=1000;
        int k;//time step
        int zmax = 20, xmax = 20, tmax=t1+1;//size of 2D TLM mesh and total simulation
                                            //time
        Double3D trans_g12(t1+3, 1, 1);//voltage impulses emerging from the TLM mesh
        string Vs="0";
        double sum; //toal voltage at the interface of TLM and PSpice
        double Vr = 0;
        trans_g12 = 0.0;

        TLM TLM1(zmax, xmax, tmax);
        Pspice_cir circuit1("Vs", "C1", "L1","input");
        TLM_Pspice interface1;
        TLM1.exitation(1.0);
        for (k = 0; k < t1; k++)
        {
                TLM1.scattering(k); //TLM scattering
                TLM1.cal_vnode(2,2); //calculate node voltage at mesh location (2,2)
                trans_g12[k+1] = TLM1.transG12();//voltage impulses emerging from TLM
                                                 //mesh
                if (k%100==0)//Display every 100 timesteps
                {
                cout << " " << endl;
                cout << "New Iteration Starts here: " << endl;
                cout << k << endl;
                }
                Vs = interface1.toPspice(trans_g12[k+1],1);

                //Update PSpice circuit file and invoke PSpice engine and
                //calculate total voltage at the TLM and PSpice interfce//
                circuit1.update(Vs.c_str());
                sum = circuit1.calculate();

                //TLM Connection and Boundary Condition///
                TLM1.connection();
                TLM1.boundary(1,-0.17157,1,Vr);
                Vr = interface1.toTLM(sum); //Calculate voltages goes back to TLM mesh
        }

        clock_t finish = clock();
        cout << endl << "Elapsed Time [sec] : " << (finish-start)/CLOCKS_PER_SEC;
```

```
        cout << endl << "Job done, hit <enter> to terminate the program.";
        cin.get();
        return 0;
}
```

B.3 Implementations for Section 5.1.3

```
//*********      Readme      ***********//
/* Purpose of the code: (TLM1+PSpice+TLM2)
          Direct one-port connection between two TEM structures (TLM1 and TLM2)
          and PSpice circuit (circuit1.cir) is between the two TEM structures

          TLM1 and TLM2 have the same size (zmax, xmax)
          PSpice circuit contains shunt connected resistor R1,capacitor C1 and
          inductor L1
Goal:     Prove direct connection obtained with the proposed algorithm has good
          agreement with MEFiSTo-2D
Procedure: 1. Create PSpice circuit file input.cir including shunt connected resistor R1,
              capacitor C1 and inductor L1
           2. circuit1 is object of class Pspice_cir; TLM1 and TLM2 are objects of class
              TLM; interface1 and interface2 are objects of class TLM_Pspice
           3. Node voltage at a specific node location in the TLM1 mesh (2,2) is saved
              to a text file node.txt
           4. Reference node voltage is saved to a text file ref.txt
           5. Create the same structure in MEFiSTo-2D and save node voltage at the same
              location and reference node voltage node_mef.txt and ref_mef.txt,
              respectively.
           6. Use matlab file myfft.m to calculate |S11| and S11 phase at the above
              mesh location (2,2) using the proposed algorithm and MEFiSTo-2D
           7. Compare results from the proposed algorithm and MEFiSTo-2D.
*/

#include "stdafx.h"
#include "Array3D.h"
#include "TLM.h"
#include "Pspice_cir.h"
#include "TLM_Pspice.h"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
        clock_t start = clock();

        int t1=1000;
        int k; //time step
        int zmax = 20, xmax = 20, tmax=t1+1;//size of 2D TLM mesh and total simulation
                                            //time
        double sum; //toal voltage at the interface of TLM and PSpice
        double vr1 = 0.0, vr2=0.0;//reflect voltages back to TLM mesh
        string VE1="0"; //Thevenin equivalent source for TLM1
        string VE2="0"; //Thevenin equivalent source for TLM2

        Double3D VEi1(t1+3,1,1), VEi2(VEi1);//voltage impulses emerging from TLM mesh
        VEi1 = VEi2 = 0.0;
```

```
        TLM TLM1(zmax, xmax, tmax);
        TLM TLM2(zmax, xmax, tmax);
        Pspice_cir circuit1("VE1", "VE2", "C1", "L1","input");
        TLM_Pspice interface1;
        TLM_Pspice interface2;
        TLM1.exitation(1.0);
        for (k = 0; k < t1; k++)
        {
                ////TLM scattering
                TLM1.scattering(k);
                TLM2.scattering(k);
                TLM1.cal_vnode(2,2); //calculate node voltage at mesh location (2,2)
                VEi1[k+1] = TLM1.transG12();
                VEi2[k+1] = TLM2.reflectG11();

                if (k%100==0)//Display every 100 timesteps
                {
                cout << " " << endl;
                cout << "New Iteration Starts here: " << endl;
                cout << k << endl;
                }

                // Update thevenin equivalent voltage VE1 and VE2 for TLM1 and TLM2
                VE1 = interface1.toPspice(VEi1[k+1],1);
                VE2 = interface2.toPspice(VEi2[k+1],1);

                //Update PSPICE circuit file and PSPICE calculation//
                circuit1.update2(VE1.c_str(),VE2.c_str());
                sum = circuit1.calculate();

                //Connection and boundary condition for TLM1 and TLM2
                TLM1.connection();
                TLM1.boundary(1,-0.17157,1,vr1);
                TLM2.connection();
                TLM2.boundary(1,vr2, 1, -0.17157);
                vr1 = interface1.toTLM(sum);//voltage impulses back to TLM1
                vr2 = interface2.toTLM(sum);//voltage impulses back to TLM2
        }

        clock_t finish = clock();
        cout << endl << "Elapse Time [sec] : " << (finish-start)/CLOCKS_PER_SEC;
        cout << endl << "Job done, hit <enter> to terminate the program.";
        cin.get();
        return 0;
}
```

B.4 Implementations for Section 5.2

```
//*********     Readme     **********//
/* Purpose of the code: (TLM1+Pspice+TLM2(G))
                Connecting one-port Johns response to PSpice by convolution

                TLM1 and TLM2 are two TEM structures with the same size (zmax, xmax)
                PSpice circuit contains shunt connected resistor R1,capacitor C1 and
```

```
                inductor L1
Goal:     Simulation results obtained witht he convolution algorithm are in good
          agreement with those produced by MEFiSTo-2D
Procedure: 1. Find Johns response G11 and G12 of TLM2
           2. Creat PSpice circuit file input.cir
           3. Set initialize condition of Vs to 0.
           4. Set initial condition of Capacitor (C) and Inductor (1) to 0.
           5. Calculate voltage impulses leaving from TLM substructure at the interface
              of TLM substructure and PSpice
           6. Invoke PSpice engine and calculate the voltage impulses from PSpice to
              TLM substructure using convolution
           7. The node voltage at TLM mesh location (2,2) is saved into a text file
              node.txt
           8. Calculate  S11  and S11 phase at the mesh location (2,2) using the
              proposed algorithm
           9. Compare S11 results with MEFiSTo and theoretical results using Matlab file
              myfft.m
*/

#include "stdafx.h"
#include "Array3D.h"
#include "TLM.h"
#include "Pspice_cir.h"
#include "TLM_Pspice.h"
#include "Conv_Pspice.h"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
        clock_t start = clock();

        int t1=1000;
        int zmax = 20, xmax = 20, tmax=t1+1;//size of 2D TLM mesh and total simulation
                                            //time
        double sum;//toal voltage at the interface of TLM and PSpice
        double vr1 = 0.0, vr2=0.0;//reflect voltages back to TLM mesh
        string VE1="0"; //Thevenin equivalent source for TLM1
        string VE2="0"; //Thevenin equivalent source for TLM2
        Double3D g11(t1+3,1,1), g12(g11);//Johns response of TLM2
        Double3D VEi1(t1+3,1,1), VEi2(VEi1); //voltage impulses emerging from TLM mesh
        Double3D Vnode(VEi1); //Node voltage
        Double3D reflect(VEi1);
        Double3D VEr2(VEi1);//voltage impulses back to TLM2

        g11=0;  g12=0;
        VEi1 = VEi2 = 0.0;
        VEr2 = 0.0;
        reflect = 0.0;

        TLM TLM1(zmax, xmax, tmax);
        Pspice_cir circuit1("VE1", "VE2", "C1", "L1","input");
        Conv_Pspice interface1;
        Conv_Pspice interface2;
```

```
//Calculate G11 and G12 of one TLM structure TLM2
TLM TLM2(zmax, xmax, tmax);
TLM2.exitation(1.0);
for (int i = 0; i < t1; i++)
{
        TLM2.scattering(i);
        g12[i+1] = TLM2.transG12();
        g11[i+1] = TLM2.reflectG11();
        TLM2.connection();
        TLM2.boundary(1,-0.17157,1,-0.17157);
}

TLM1.exitation(1.0);
for (int k = 0; k < t1; k++)
{
        ////TLM scattering
        TLM1.scattering(k);
        TLM1.cal_vnode(2,2); //calculate node voltage at mesh location (2,2)
        VEi1[k+1] = TLM1.transG12();

        if (k%100==0)//Display every 100 timesteps
        {
        cout << " " << endl;
        cout << "New Iteration Starts here: " << endl;
        cout << k << endl;
        }

        // Update thevenin equivalent voltage VE1 and VE2 for TLM1 and TLM2
        VE1 = interface1.toPspice(VEi1[k+1],1);
        VE2 = interface2.toPspice(VEi2[k+1],1);

        //Update PSPICE circuit file and PSPICE calculation
        circuit1.update2(VE1.c_str(),VE2.c_str());
        sum = circuit1.calculate();
        VEr2[k+2] = interface2.toTLM(sum);

        reflect = interface2.Convolve(VEr2[k+1],g11,t1+3,k); //find voltage from
                                                //TLM1 to PSpice by convolution
        VEi2[k+1]=reflect[k+1];

        //Connection and boundary condition for TLM1 and TLM2
        TLM1.connection();
        TLM1.boundary(1,-0.17157,1,vr1);
        vr1 = interface1.toTLM(sum);
}

clock_t finish = clock();
cout << endl << "Elapse Time [sec] : " << (finish-start)/CLOCKS_PER_SEC;
cout << endl << "Job done, hit <enter> to terminate the program.";
cin.get();
return 0;
}
```

B.5 Implementations for Section 5.3

```
//*********      Readme      **********//
/* Purpose of the code: (TLM1+Pspice1+TLM2+Pspice2)
          Direct two-port connection between TLM substructures (TLM1 and TLM2) with
          PSpice circuits (pspice1.cir and pspice2.cir)

          TLM1 and TLM2 are two TEM structures with the same size (zmax, xmax)
          PSpice circuit pspice1.cir and pspice2.cir contain shunt connected resistor
          R1, capacitor C1 and inductor L1
Goal:     Prove direction connection obtained with the proposed algorithm has good
          agreement with MEFiSTo-2D
Procedure: 1. Create two PSpice circuit files: pspice1.cir and pspice2.cir: both
              circuits includes shunt connected resistor, capacitor and inductor
           2. Circuit1 and circuit2 are objects of class Pspice_cir;
              TLM1 and TLM2 are objects of class TLM;
           3. Node voltage at a specific node location in the TLM1 mesh (2,2) is saved
              to a text file node.txt
           4. Reference node voltage is saved to a text file ref.txt
           5. Create the same structure in MEFiSTo-2D and save node voltage at the same
              location and reference node voltage node_mef.txt and ref_mef.txt,
              espectively.
           6. Use matlab file myfft.m to calculate |S11| and S11 phase at the above mesh
              location (2,2) using the proposed algorithm and MEFiSTo-2D
           7. Compare results from the proposed algorithm and MEFiSTo-2D.
*/
#include "stdafx.h"
#include "Array3D.h"
#include "TLM.h"
#include "Pspice_cir.h"
#include "TLM_Pspice.h"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
        clock_t start = clock();

        int t1=1000;
        int k;
        int zmax = 20, xmax = 20, tmax=t1+1;//size of 2D TLM mesh and total simulation
                                            //time
        double sum1, sum2; //total voltages at two TLM and PSpice interfaces
        double vr1 = 0.0, vr2=0.0, vr3=0.0;
        string VE1="0"; //Thevenin equivalent source for TLM1
        string VE2="0"; //Thevenin equivalent source for TLM2
        string VE3="0"; //Thevenin equivalent source for TLM3
        Double3D VEi1(t1+3,1,1), VEi2(VEi1), VEi3(VEi1), Vnode(VEi1);
        VEi1 = VEi2 = VEi3 = 0.0;

        TLM TLM1(zmax, xmax, tmax);
        TLM TLM2(zmax, xmax, tmax);//Create two objects TLM1 and TLM2 of class TLM

        //Create two objects circuit1 and circuit2 of class Pspice_cir
        //Two circuit files: pspice1.cir and pspice2.cir
        Pspice_cir circuit1("VE1", "VE2", "C1", "L1","pspice1");
```

```
        Pspice_cir circuit2("VE3", "C2", "L2","pspice2");
        TLM_Pspice interface1;
        TLM_Pspice interface2;
        TLM_Pspice interface3; //Create three objects of class TLM_Pspice

        TLM1.exitation(1.0);
        for (k = 0; k < t1; k++)
        {
                //Scattering for TLM1 and TLM2
                TLM1.scattering(k);
                TLM2.scattering(k);
                TLM1.cal_vnode(2,2);
                VEi1[k+1] = TLM1.transG12();
                VEi2[k+1] = TLM2.reflectG11();
                VEi3[k+1] = TLM2.transG12();

                if (k%100==0)//Display every 100 timesteps
                {
                cout << " " << endl;
                cout << "New Iteration Starts here: " << endl;
                cout << k << endl;
                }

                VE1 = interface1.toPspice(VEi1[k+1],1);
                VE2 = interface2.toPspice(VEi2[k+1],1);
                VE3 = interface3.toPspice(VEi3[k+1],1);

                //Update PSPICE circuit file and PSPICE calculation//
                circuit1.update2(VE1.c_str(),VE2.c_str());
                sum1 = circuit1.calculate();
                circuit2.update(VE3.c_str());
                sum2 = circuit2.calculate();

                //Connection and boundary conditions for TLM1 and TLM2
                TLM1.connection();
                TLM1.boundary(1,-0.17157,1,vr1);
                TLM2.connection();
                TLM2.boundary(1,vr2, 1, vr3);
                vr1 = interface1.toTLM(sum1);//voltage impulses back to TLM1
                vr2 = interface2.toTLM(sum1);//voltage impulses back to TLM2
                vr3 = interface3.toTLM(sum2);//voltage impulses back to TLM3
        }

        clock_t finish = clock();
        cout << endl << "Elapse Time [sec] : " << (finish-start)/CLOCKS_PER_SEC;
        cout << endl << "Job done, hit <enter> to terminate the program.";
        cin.get();
        return 0;
}
```

B.6 Implementations for Section 5.4.1

```
//*********      Readme      **********//
/* Purpose of the code: (TLM1+Pspice1+TLM2(G)+Pspice2)
          two-port connection between TLM substructures (TLM1 and TLM2) with
          PSpice circuits (pspice1.cir and pspice2.cir) using convolution

          TLM1 and TLM2 are two TEM structures with the same size (zmax, xmax)
          PSpice circuit pspice1.cir and pspice2.cir contain shunt connected resistor
          R1, capacitor C1 and inductor L1
Goal:     Prove connection obtained with the proposed algorithm using convolution has
          good agreement with MEFiSTo-2D
Procedure: 1. Create two PSpice circuit files: pspice1.cir and pspice2.cir;
             both circuits includes shunt connected resistor, capacitor and inductor
          2. Calculate Johns response of TLM2 G11 and G12
          3. Circuit1 and circuit2 are objects of class Pspice_cir;
             TLM1 and TLM2 are objects of class TLM;
          4. Node voltage at a specific node location in the TLM1 mesh (2,2) is
             saved to a text file node.txt
          5. Reference node voltage is saved to a text file ref.txt
          6. Create the same structure in MEFiSTo-2D and save node voltage at the
             same location and reference node voltage node_mef.txt and ref_mef.txt,
             respectively.
          7. Use matlab file myfft.m to calculate |S11| and S11 phase at the above
             mesh location (2,2) using the proposed algorithm and MEFiSTo-2D
          8. Compare results from the proposed algorithm and MEFiSTo-2D.
*/

#include "stdafx.h"
#include "Array3D.h"
#include "TLM.h"
#include "Pspice_cir.h"
#include "TLM_Pspice.h"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
        clock_t start = clock();

        int t1=1000;
        int zmax = 20, xmax = 20, tmax=t1+1;//size of 2D TLM mesh and total simulation
                                            //time
        double sum1, sum2;//total voltage at the TLM and PSpice interface
        double vr1 = 0.0, vr2=0.0, vr3=0.0;
        string VE1="0"; //Thevenin equivalent source for TLM1
        string VE2="0"; //Thevenin equivalent source for TLM2
        string VE3="0"; //Thevenin equivalent source for TLM3
        Double3D VEi1(t1+3,1,1), VEi2(VEi1), VEi3(VEi1), Vnode(VEi1);
        Double3D VEr1(VEi1), VEr2(VEi1), VEr3(VEi1);
        Double3D reflect1(VEi1), trans1(VEi1), trans1b(VEi1), trans2(VEi1);
        Double3D ng11(VEi1), ng12(VEi1);
        VEi1 = VEi2 = VEi3 = 0.0;//voltage impulses to PSpice
        VEr1 = VEr2 = VEr3 = 0.0;//voltage impulses to TLM mesh
        reflect1 = trans1 = trans1b = trans2 = 0.0;
        ng11 = ng12 = 0.0;//Johns response G11 and G12
```

```
TLM TLM1(zmax, xmax, tmax);
TLM TLM2(zmax, xmax, tmax);//Create two objects TLM1 and TLM2 of class TLM

//Create two objects circuit1 and circuit2 of class Pspice_cir
//Two circuit files: pspice1.cir and pspice2.cir
Pspice_cir circuit1("VE1", "VE2", "C1", "L1","pspice1");
Pspice_cir circuit2("VE3", "C2", "L2","pspice2");
TLM_Pspice interface1;
TLM_Pspice interface2;
TLM_Pspice interface3; //Create three objects of class TLM_Pspice

 // Get Johns response G11 and G12 of TLM2
TLM2.exitation(1.0);
for (int k = 0; k < t1; k++)
{
        TLM2.scattering(k);
        ng11[k+1] = TLM2.reflectG11();
        ng12[k+1] = TLM2.transG12();
        TLM2.connection();
        TLM2.boundary(1, 0, 1, 0);
}

TLM1.exitation(1.0);
for (int k = 0; k < t1; k++)
{
        //Scattering for TLM1 and TLM2
        TLM1.scattering(k);
        TLM1.cal_vnode(2,2);
        VEi1[k+1] = TLM1.transG12();

        if (k%100==0)//Display every 100 timesteps
        {
        cout << " " << endl;
        cout << "New Iteration Starts here: " << endl;
        cout << k << endl;
        }

        VE1 = interface1.toPspice(VEi1[k+1],1);
        VE2 = interface2.toPspice(VEi2[k],1);

        //Update PSPICE circuit file//
        circuit1.update2(VE1.c_str(),VE2.c_str());
        sum1 = circuit1.calculate();
        VEr1[k+2] = interface1.toTLM(sum1);


        VEr2[k+2] = interface2.toTLM(sum1);
        for (int n=1; n < t1-k; n++)
        {
                //reflect1 = VEr2 * G11
                reflect1[k+n]=VEr2[k+1]*ng11[n]+reflect1[k+n];
                //trans1 = VEr2 * G21
                trans1[k+n]=VEr2[k+1]*ng12[n]+trans1[k+n];
        }
```

```
            VEi3[k]=trans1[k]+trans1b[k];//****VEi3 = trans1 + trans1b
            VE3 = interface3.toPspice(VEi3[k],1);

            //Connection and boundary conditions for TLM1 and TLM2
            TLM1.connection();
            TLM1.boundary(1,-0.17157,1,VEr1[k+1]);

            circuit2.update(VE3.c_str());
            sum2 = circuit2.calculate();
            VEr3[k+2] = interface3.toTLM(sum2);

            for (int n=1; n < t1-k; n++)
            {
                    //trans1b = VEr3 * G22
                    trans1b[k+n]=VEr3[k+1]*ng11[n]+trans1b[k+n];
                    //trans2 = VEr3 * G12
                    trans2[k+n]=VEr3[k+1]*ng12[n]+trans2[k+n];
            }
            VEi2[k+1]=reflect1[k+1]+trans2[k+1];
    }

    clock_t finish = clock();
    cout << endl << "Elapse Time [sec] : " << (finish-start)/CLOCKS_PER_SEC;
    cout << endl << "Job done, hit <enter> to terminate the program.";
    cin.get();
    return 0;
}
```

B.7 Implementations for Section 5.4.2

```
//*********      Readme      ***********//
/* Purpose of the code: (TLM1+TLM2(G)+Pspice2)
            Connection of two-port Johns responses to PSpice by convolution

            TLM1 and TLM2 are two TEM structures with the same size (zmax, xmax)
            PSpice circuit pspice2.cir contain shunt connected resistor R1, capacitor
            C1 and inductor L1
Goal:       Prove connection obtained with the proposed convolution algorithm has good
            agreement with MEFiSTo-2D(with no convolution)
Procedure: 1. Create PSpice circuit file pspice2.cir including shunt connected resistor,
              capacitor and inductor
           2. Johns reponse G11 and G12 of TLM2 is obtained first in MEFiSTo-2D
           3. Circuit2 is object of class Pspice_cir; TLM1 is object of class TLM;
              interface1 is object of class ConvPspice
           4. Node voltage at a specific node location in the TLM1 mesh (2,2) is saved
              to a text file node.txt
           5. Reference node voltage is saved to a text file ref.txt
           6. Create the same structure in MEFiSTo-2D and save node voltage at the
              same location and reference node voltage node_mef.txt and ref_mef.txt,
              respectively.
           7. Use matlab file myfft.m to calculate |S11| and S11 phase at the above
              mesh location (2,2) using the proposed algorithm and MEFiSTo-2D
           8. Compare results from the proposed algorithm and MEFiSTo-2D.
*/
```

```
#include "stdafx.h"
#include "Array3D.h"
#include "TLM.h"
#include "Pspice_cir.h"
#include "TLM_Pspice.h"
#include "Conv_Pspice.h"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
	clock_t start = clock();

	int t1=1000;
	int t2=t1;
	int zmax = 20, xmax = 20, tmax=t1+1;//size of 2D TLM mesh and total simulation
	                                    //time
	double sum;//total voltage at the TLM and PSpice interface

	string VE3="0";//thevenin equivalent source for TLM2
	Double3D VEi1(t1+3,1,1), VEi2(VEi1), VEi3(VEi1), VEr3(VEi1);
	Double3D reflect1(VEi1), trans1(VEi1), trans1b(VEi1), trans2(VEi1);
	Double3D g11(VEi1), g21(VEi1);//Johns response G11 and G12 of TLM2
	VEi1 = VEi2 = VEi3 = VEr3 = 0.0;
	reflect1 = trans1 = trans1b = trans2 = 0.0;
	g11 = g21 = 0.0;

	////////////////     Get G11 of TLM2      ////////////////////
	ifstream infile_g11 ("ref_g11.txt");
	string line_g11;
	double m1, m2, m3;
	int index=0;
	if(infile_g11)
	{
		do
		{
			getline (infile_g11, line_g11);
			stringstream (line_g11)>> m1 >> m2 >> m3;
			g11[index]=m3;
			index=index+1;
		}while (index!=1001);
	}
	infile_g11.close();

	////////////////      Get G21 of TLM2         ////////////////////
	ifstream infile_g21 ("ref_g21.txt");
	string line_g21;
	double mm1, mm2, mm3;
	index=0;
	if(infile_g21)
	{
		do
		{
			getline (infile_g21, line_g21);
			stringstream (line_g21)>> mm1 >> mm2 >> mm3;
			g21[index+1]=mm3;
```

```
                index=index+1;
        }while (index!=1001);
}
infile_g21.close();

TLM TLM1(zmax, xmax, tmax);  //Create one object TLM1 of class TLM

//Create one object circuit2 of class Pspice_cir
//The circuit file: pspice2.cir
Pspice_cir circuit2("VE3", "C2", "L2","pspice2");
Conv_Pspice interface1; //Create one object interface1 of class Conv_Pspice

TLM1.exitation(1.0);
for (int k=0; k<t1; k++)
{
        TLM1.scattering(k);
        TLM1.cal_vnode(2,2);
        VEi1[k+1]=TLM1.transG12();

        if (k%100==0)//Display every 100 timesteps
        {
        cout << " " << endl;
        cout << "New Iteration Starts here: " << endl;
        cout << k << endl;
        }

        for (int n=1; n < t1-k; n++)
        {
                //reflect1 = VEr2 * G11
                reflect1[k+n]=VEi1[k+1]*g11[n]+reflect1[k+n];
                //trans1 = VEr2 * G21
                trans1[k+n]=VEi1[k+1]*g21[n]+trans1[k+n];
        }
        VEi3[k]=trans1[k]+trans1b[k];//VEi3 = trans1 + trans1b
        VE3 = interface1.toPspice(VEi3[k],1); //Update VE3

        //Connection and boundary condition for TLM1
        TLM1.connection();
        TLM1.boundary(1,-0.17157,1,VEi2[k]);

        circuit2.update(VE3.c_str());
        sum = circuit2.calculate(); //pspice2.cir calculation
        VEr3[k+2] = interface1.toTLM(sum);

        for (int n=1; n < t1-k; n++)
        {
                //trans1b = VEr3 * G22
                trans1b[k+n]=VEr3[k+1]*g11[n]+trans1b[k+n];
                //trans2 = VEr3 * G12
                trans2[k+n]=VEr3[k+1]*g21[n]+trans2[k+n];
        }
VEi2[k+1]=reflect1[k+1]+trans2[k+1]; // VEi2 = reflect1 + trans2
}

clock_t finish = clock();
```

```
        cout << endl << "Elapse Time [sec] : " << (finish-start)/CLOCKS_PER_SEC;
        cout << endl << "Job done, hit <enter> to terminate the program.";
        cin.get();
        return 0;
}
```

B.8 Implementations for Section 6.1

```
//*********      Readme      **********//
/*Validation Example1:
Single stub tuner: Connection of two-port Johns responses to PSpice by convolution
                   (TLM1+TLM2(G)+Pspice2)
                   Extract TEM mode when higher order modes are also present in the
                   structure
Goal:       Prove connection obtained with the proposed convolution algorithm has good
            agreement with MEFiSTo-2D(with no convolution)
Procedure: 1. Create a PSpice circuit file circuit2.cir
           2. Divide the whole structure into 3 parts: 2 TLM structures, TLM1 and TLM2,
              and PSpice circuit
           3. In MEFiSTo-2D, put 5 probes at the input and output port of TLM2 to
              calculate its Johns response G11 and G21
           4. Create object TLM1 of class TLM and object circuit2 of class Pspice_cir
           5. Connect TLM1, TLM2 and circuit2 using convolution
           6. Save node voltage at a specific mesh location to text file node.txt
           7. Reference node voltage is saved to a text file ref.txt
           8. Create the same structure in MEFiSTo-2D and save node voltage at the same
              location and reference node voltage node_mef.txt and ref_mef.txt,
              respectively.
           9. Use matlab file myfft.m to calculate |S11| and S11 phase at the above mesh
              location using the proposed algorithm and MEFiSTo-2D
           10. Compare results from the proposed algorithm and MEFiSTo-2D.
*/

#include "stdafx.h"
#include "Array3D.h"
#include "TLM.h"
#include "Pspice_cir.h"
#include "TLM_Pspice.h"
#include "Conv_Pspice.h"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
        clock_t start = clock();

        int t1=1000;
        int t2=t1;
        int zmax = 30, xmax = 5, tmax=t1+1;//size of 2D TLM mesh and total simulation
                                           //time
        double sum;//total voltage at the TLM and PSpice interface
        double h=0.000663605;// Height of 2D TLM structure

        string VE3="0";//Thevenin equivalent source for TLM2
        Double3D VEi1(t1+3,1,1), VEi2(VEi1), VEi3(VEi1), VEr3(VEi1);
```

```
Double3D reflect1(VEi1), trans1(VEi1), trans1b(VEi1), trans2(VEi1);
Double3D g11(10005,1,1), g21(10005,1,1);
VEi1 = VEi2 = VEi3 = VEr3 = 0.0;
reflect1 = trans1 = trans1b = trans2 = 0.0;
g11 = g21 = 0.0;//Johns response G11 and G21 for TLM2

ifstream infile_pin1 ("pin1.txt");
ifstream infile_pin2 ("pin2.txt");
ifstream infile_pin3 ("pin3.txt");
ifstream infile_pin4 ("pin4.txt");
ifstream infile_pin5 ("pin5.txt");
ofstream outfile_g11 ("g11.txt");
string line_g11;
double m1, m2, m3,sum_in;

////  Get G11 of TLM2 structure:
int index=0;
if(infile_pin1&&infile_pin2&&infile_pin3&&infile_pin4&&infile_pin5)
{
        do
        {
                sum_in=0;
                getline (infile_pin1, line_g11);
                stringstream (line_g11)>> m1 >> m2 >> m3;
                sum_in=sum_in+m3;
                getline (infile_pin2, line_g11);
                stringstream (line_g11)>> m1 >> m2 >> m3;
                sum_in=sum_in+m3;
                getline (infile_pin3, line_g11);
                stringstream (line_g11)>> m1 >> m2 >> m3;
                sum_in=sum_in+m3;
                getline (infile_pin4, line_g11);
                stringstream (line_g11)>> m1 >> m2 >> m3;
                sum_in=sum_in+m3;
                getline (infile_pin5, line_g11);
                stringstream (line_g11)>> m1 >> m2 >> m3;
                sum_in=sum_in+m3;
                if (index==0)
                {
                        g11[index+1]=sum_in/5-1;
                }
                else g11[index+1]=sum_in/5;
                outfile_g11 << g11[index]<< endl;
                index=index+1;
        }while (index!=10004);
}
infile_pin1.close();
infile_pin2.close();
infile_pin3.close();
infile_pin4.close();
infile_pin5.close();
outfile_g11.close();

//////////  Get G21 of TLM2 structure:
ifstream infile_pout1 ("pout1.txt");
```

```
ifstream infile_pout2 ("pout2.txt");
ifstream infile_pout3 ("pout3.txt");
ifstream infile_pout4 ("pout4.txt");
ifstream infile_pout5 ("pout5.txt");
ofstream outfile_g21 ("g21.txt");
string line_g21;
double sum_out;
index=0;
if(infile_pout1&&infile_pout2&&infile_pout3&&infile_pout4&&infile_pout5)
{
        do
        {
                sum_out=0;
                getline (infile_pout1, line_g21);
                stringstream (line_g21)>> m1 >> m2 >> m3;
                sum_out=sum_out+m3;
                getline (infile_pout2, line_g21);
                stringstream (line_g21)>> m1 >> m2 >> m3;
                sum_out=sum_out+m3;
                getline (infile_pout3, line_g21);
                stringstream (line_g21)>> m1 >> m2 >> m3;
                sum_out=sum_out+m3;
                getline (infile_pout4, line_g21);
                stringstream (line_g21)>> m1 >> m2 >> m3;
                sum_out=sum_out+m3;
                getline (infile_pout5, line_g21);
                stringstream (line_g21)>> m1 >> m2 >> m3;
                sum_out=sum_out+m3;
                g21[index+1]=sum_out/5;
                outfile_g21 << g21[index]<< endl;
                index=index+1;
                }while (index!=10004);
}
infile_pout1.close();
infile_pout2.close();
infile_pout3.close();
infile_pout4.close();
infile_pout5.close();
outfile_g21.close();

TLM TLM1(zmax, xmax, tmax);  //Create one object TLM1 of class TLM

//Create one object of class Pspice_cir
//The circuit file: pspice2.cir
Pspice_cir circuit2("VE3", "C2","pspice2");
Conv_Pspice interface1; //Create one object interface1 of class Conv_Pspice

TLM1.exitation(1.0);
for (int k=0; k<t1; k++)
{
        TLM1.scattering(k);//TLM1 scattering
        TLM1.cal_vnode(1,(xmax-1)/2);
        VEi1[k+1]=TLM1.transG12();//voltage impulses emerging from TLM1

        if (k%100==0)//Display every 100 timesteps
```

```
                {
                cout << " " << endl;
                cout << "New Iteration Starts here: " << endl;
                cout << k << endl;
                }

                for (int n=1; n < t2-k; n++)
                {
                        //reflect1 = VEi1 * G11
                        reflect1[k+n]=VEi1[k+1]*g11[n]+reflect1[k+n];
                        //trans1 = VEi1 * G21
                        trans1[k+n]=VEi1[k+1]*g21[n]+trans1[k+n];
                }
                VEi3[k]=trans1[k]+trans1b[k];//VEi3 = trans1 + trans1b
                VE3 = interface1.toPspice(VEi3[k],h); //Update VE3

                //Connection and boundary condition for TLM1
                TLM1.connection();
                TLM1.boundary(1,-0.17157,1,VEi2[k]);

                circuit2.update(VE3.c_str());
                sum = circuit2.calculate2(); //pspice2.cir calculation
                VEr3[k+2] = interface1.toTLM(sum);

                for (int n=1; n < t2-k; n++)
                {
                        //trans1b = VEr3 * G22
                        trans1b[k+n]=VEr3[k+1]*g11[n]+trans1b[k+n];
                        //trans2 = VEr3 * G21
                        trans2[k+n]=VEr3[k+1]*g21[n]+trans2[k+n];
                }
        VEi2[k+1]=reflect1[k+1]+trans2[k+1]; // VEi2 = reflect1 + trans2
        }

        clock_t finish = clock();
        cout << endl << "Elapse Time [sec] : " << (finish-start)/CLOCKS_PER_SEC;
        cout << endl << "Job done, hit <enter> to terminate the program.";
        cin.get();
        return 0;
}
```

B.9 Implementations for Section 6.2

```
//*********     Readme     ***********//
/* Validation Example2: Inductive iris in waveguide (TE10 mode)
     Extract TEM mode when higher order modes are also present in the structure

Procedure:
        1. A section of waveguide can be divided into two subsections TLM1 and TLM2.
        2. In MEFiSTo, put 11 probes at one port of TLM2 subsection to get Johns
           response G11
        3. Find a simple mode profile at the interface
        4. Reconnect TLM1 with TLM2 using multi-modal convolution
        5. Use the simple mode profile at the interface to simplify the convoulution
```

```
        procedure
    6. Save node voltage at a specific mesh location to text file node.txt
    7. Reference node voltage is saved to a text file ref.txt
    8. Create the same structure in MEFiSTo-2D and save node voltage at the same
       location and reference node voltage node_mef.txt and ref_mef.txt,respectively.
    9. Use matlab file myfft.m to calculate  S11  and S11 phase at the above mesh
       location using the proposed algorithm and MEFiSTo-2D
    10. Compare results from the proposed algorithm and MEFiSTo-2D.
*/
//////////////////Done: ////////////////////////////
#include "stdafx.h"
#include "Array3D.h"
#include "TLM.h"
#include "TLM_TE.h"
#include "TLM_Pspice.h"
#include "Conv_Pspice.h"

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
        clock_t start = clock();

        int t1=10000;
        int zmax = 10, xmax = 11, tmax=t1+1;//size of 2D TLM mesh and total simulation
                                            //time
        Double3D VEi1(t1+3,1,1), reflect1(VEi1);//voltage impulses emerging from TLM
                                                //substructure
        Double3D g11(tmax+5,1,1);
        double sum;

        VEi1 = 0; reflect1 = 0; g11 = 0;

        //a). Using probes to get the voltages coming out from each branch in
        //       the John's Matrix structure cross section.
        //b). Calculate the TE10 foundamental mode for John's Matrix which are
        //       used to connect with the rest structures.
        //c). Save the results to g11.txt
        TLM_TE TLM2(zmax, xmax, tmax);
        TLM_TE TLM1(zmax, xmax, tmax);

        g11 = TLM1.cal_g11();//G11 of TLM2

        TLM1.exitation(11);//Excite TE10 mode

        for (int k = 0; k < t1; k++)
        {
                TLM1.scattering(k);//scattering for TLM1
                TLM1.cal_vnode();//calculate node voltage at a certain location
                sum = TLM1.cal_sum();
                VEi1[k+1]=sum*2/11;

                if (k%100==0)//Display every 100 timesteps
                {
                cout << " " << endl;
```

```
            cout << "New Iteration Starts here: " << endl;
            cout << k << endl;
            }

            for (int n=1; n < t1-k; n++)
            {
                    //reflect1 = VEr2 * G11
                    reflect1[k+n]=VEi1[k+1]*g11[n]+reflect1[k+n];
            }

            //Connection and boundary condition
            TLM1.connection();
            TLM1.boundary(-1,0,-1,reflect1[k]);
    }

    clock_t finish = clock();
    cout << endl << "Elapsed Time [sec] : " << (finish-start)/CLOCKS_PER_SEC;
    cout << endl << "Job done, hit <enter> to terminate the program.";
    cin.get();
    return 0;
}
```