035067-10-T

USERS MANUAL FOR FSSEIGER / FSSBUILD

D. R. Wilton and D. R. Jackson

April 1998

35067-10-T = RL-2490

PROJECT INFORMATION

PROJECT TITLE:	Hybrid Finite Element Design Codes for the SERAT Array
REPORT TITLE:	Users Manual for FSSEiger / FSSBuild
U-M REPORT No.:	035067-9-T
CONTRACT START DATE:	October 1996
END DATE:	September 1998
DATE:	April 5, 1998

SPONSOR:

Roland Gilbert
SANDERS, INC, A Lockheed Martin Co.
MER 24-1583
PO Box 868
Nashua, NH 030601-0868
Phone: (603) 885-5861
Email: RGILBERT@mailgw.sanders.lockheed.com

SPONSOR CONTRACT No.: P.O. QP2047

U-M PRINCIPAL INVESTIGATOR:

John L. Volakis
EECS Dept.
University of Michigan
1301 Beal Ave
Ann Arbor, MI 48109-2122
Phone: (313) 764-0500 FAX: (313) 747-2106
volakis@umich.edu
http://www-personal.engin.umich.edu/~volakis/

CONTRIBUTORS TO THIS REPORT: D. R. Wilton (Univ. of Houston) and D. R. Jackson (Univ. of Houston)

FSSEIGER / FSSBUILD

USER'S MANUAL

Applied Electromagnetics Laboratory
Department of Electrical and Computer Engineering
University of Houston

Donald R. Wilton
David R. Jackson

April 1, 1998

FSSEIGER / FSSBUILD

USER'S MANUAL

TABLE OF CONTENTS

I. INTRODUCTION

This user's manual describes the use of two sets of Fortran 90 codes, FSSEIGER and FSSBUILD, for the analysis of radiation and scattering from periodic structures in layered media.

FSSEIGER is a suite of Fortran 90 codes for the analysis of radiation and scattering from periodic structures in a layered media. The FSSEIGER code consists of an existing general purpose code, EIGER, together with an application module which extends its postprocessing capabilities to provide parameters of interest to designers of frequency selective surfaces (FSS) and phased arrays. EIGER has the capability to treat problems of radiation and scattering by structures in either periodic or nonperiodic layered media. It implements a mixed potential integral equation (MPIE) formulation using subdomain basis functions, thus allowing flexible modeling of structures having arbitrary shape, including both planar and nonplanar objects, inside a general layered media. Although the current implementation of the FSSEIGER and FSSBUILD combination employs only a relatively small subset of the features of EIGER, it does inherit EIGER's flexible modeling characteristics. EIGER also provides the capability of extending the suite's usefulness in the future by employing other EIGER modeling features.

To meet the specific needs of current FSS and phased-array design requirements, a geometry-building user interface called FSSBUILD has been created. FSSBUILD acts as a pre-processor to the FSSEIGER set of codes. The FSSBUILD codes allow the user to easily construct the necessary geometry and problem specification file (the *.eig file) which is used as input to the FSSEIGER codes. Although the EIGER code can handle quite general geometries, the FSSBUILD set of codes is restricted to geometry generation using four types of planar elements only. These four types of planar elements are:

- Rectangular-shaped metallic patches or dipoles
- Cross-shaped metallic patches or dipoles
- Rectangular-shaped slot in a ground plane
- Cross-shaped slot in a ground plane

Each element type is constructed by FSSBUILD using triangular-element subdomains. (although EIGER allows for planar rectangular or wire element subdomains in periodic structure modeling, these capabilities have not been included in the geometry-building set of codes.) The complete geometry may consist of arbitrary numbers of FSS arrays, each of which consists of a periodic array of one of the above four element types. Each FSS array may reside inside a dielectric layer, or at the interface between dielectric layers. The number of dielectric layers is arbitrary, and is independent of the number of FSS arrays. The dielectric layers may also have arbitrary permittivity, permeability, and loss tangent. For metallic array elements, the codes allow for lumped loads to be placed across an arbitrary number of triangular edges.

For analyzing radiation problems, FSSBUILD allows the user to define (periodic) voltage sources across an arbitrary number of triangular edges. For scattering problems, the user defines the incident angle and the polarization of the incident wave. In either case, a scan range may be selected to allow for multiple scan angles or angles of incidence. A range of frequencies may also be selected.

EIGER generally outputs only basic information, such as a list of the equivalent currents associated with the basis functions on the object (electric currents for metallic objects and equivalent magnetic currents for apertures). In FSS and phased-array applications, EIGER instead calls the FSSEIGER postprocessor application module, passing the equivalent currents to it, which are then output to the user and also used to generate parameters of interest for periodic structures. This includes, for example, the calculation of input impedance and element patterns for radiation problems, and reflection and transmission coefficients for scattering problems.

A complete listing of the modeling capabilities of the FSSBUILD, EIGER, and FSSEIGER codes is summarized in Table 1. A summary of the output parameters calculated by FSSEIGER is given in Table 2.

In the following sections, the use of the FSSBUILD and FSSEIGER codes will be described in detail. Most of the discussion will focus on FSSBUILD, since this set of codes acts as the user interface to FSSEIGER. In most cases the user will not require an understanding of the *.eig file that is created by FSSBUILD, which is the input file to FSSEIGER. In this sense, the FSSEIGER set of codes is essentially transparent to the user.

The precise compilation, linking, and running of the FSSBUILD and FSSEIGER codes will depend on the platform and version of Fortran that is being used. In Appendix A, a brief discussion of how the codes would typically be set up and executed using Digital Visual Fortran is given.

TABLE 1

Summary of Modeling Capabilities

FEATURES	FSSBUILD	EIGER	FSSEIGER
Excitations:			
Plane wave	√	√	√
Voltage source	√	√	√
Element Geometries:			
Planar dipole	√	√	√
Planar cross	√	√	√
Planar slot	√	√	√
Planar cross slot	√	√	√
Subdomains:			
Triangular	√	√	√
Rectangular		√	√
Wire		√	√
Wire/triangle junctions		√	√
Boundary Conditions:			
EFIE	√	√	√
MFIE	(slots only)	√	√
CFIE		√	√
Dielectric (PMCHW)		√	√
Loading:			
Lumped impedance	√	√	√
Surface impedance		√	√
Green's Functions:			
Homogeneous nonperiodic		√	
Layered media:			
Planar periodic (skewed)	√	√	√
Noncommensurate periodic		√	√
Nonperiodic		√	

TABLE 2

Parameters Computed by FSSEIGER

Plane Wave Excitation:	
Equivalent currents	√
Reflection coefficient	√
Transmission coefficient	√
Voltage Excitation:	
Equivalent currents	√
Input power per element	√
Active impedance and admittance	√
Array element pattern	√

II. GEOMETRY INFORMATION

As mentioned in the previous section, FSSBUILD allows the user to generate a combination of FSS arrays consisting of four element types, with each FSS or array structure arbitrarily located within a dielectric layered structure. The geometry information for the layers and the FSS elements is discussed separately in the following subsections.

A. LAYER INFORMATION

The geometry of the layered structure is shown below in Fig. 1.

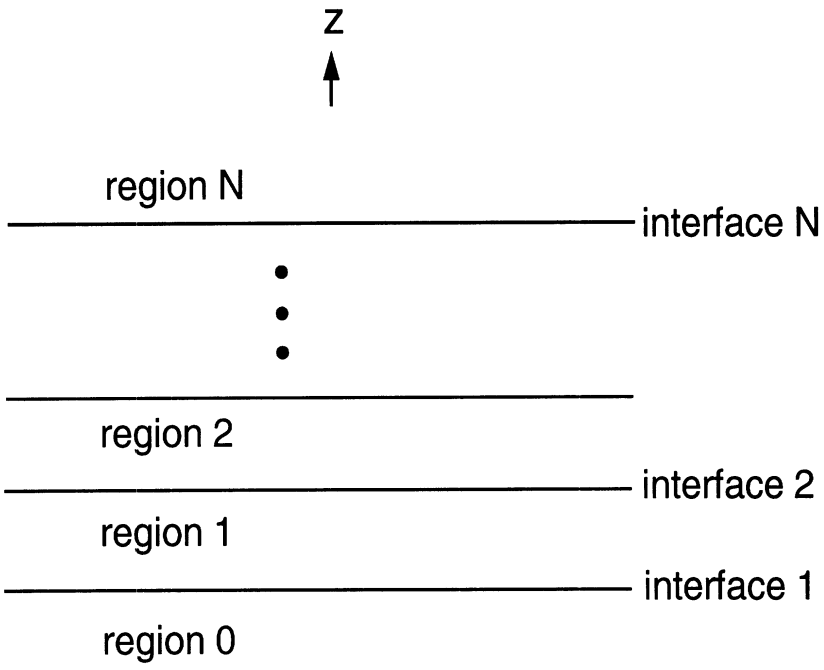

Figure 1. Geometry of the layered structure.

In the labeling scheme used by FSSBUILD and FSSEIGER, the lowest region is a half-space designated as region number 0. Each dielectric layer is then labeled with an increasing number in the positive z direction, with the last layer being layer number N. The half-space above the structure is the final layer N. The total number of dielectric layers is then N-1 (1,2,...N-1), and the total number of interfaces is N (1,2,...N). Each region (half-space of layer) has its own

complex permittivity and permeability. The layers 1,2,...*N-1* also have arbitrary independent thicknesses.

Any ground planes present are also considered to be interfaces. For example, the bottom interface number 1, between region 0 (lower half-space) and region 1 (the first layer) may be a ground plane. If an interface is a ground plane, then it may have slot elements in it (one of the two basic slot array elements mentioned previously).

The number of separate FSS arrays does not have to coincide with the number of interfaces. An FSS array may lie either at an interface or interior to any dielectric region at an arbitrary position *z*.

B. ARRAY ELEMENT INFORMATION

As mentioned, FSSBUILD allows the user the choice of four basic elements from which the periodic FSS arrays may be constructed. These are the metallic rectangular patch, the metallic cross, the rectangular slot, and the cross-shaped slot. The slot elements are the exact complements of the metallic elements (metal replaced by aperture and aperture replaced by metal). Each separate FSS array may be constructed from any one of these four element types. The four element shapes are shown in Fig. 2 and are described individually in more detail below.

Rectangular patch elements

The rectangular elements shown in Figs. 2 (a) and (b) may have arbitrary length *w* along the *x* axis and height *h* along the *y* axis. There is no restriction that the length must be larger than the height, so the patch may be square in shape, or rectangular with the long dimension aligned with either the *x* or *y* axis.

The user also has the option of specifying a general planar quadrilateral that is determined from the specified locations of the four corners, as shown in Fig. 2c. The quadrilateral may be a

rectangle that is rotated with respect to the *x*-*y* axes, or it may be a general quadrilateral with non-parallel edges.

Cross elements

Two additional element types that may be specified are the metallic cross and the cross-shaped slot in a ground plane, shown in Figs. 3 (a) and (b), respectively. The cross elements are assumed to be symmetric, so that the width and length of each arm is the same, with the two arms intersecting at the middle. The arms of the cross are assumed to be aligned with the *x* and *y* axes.

For any of the four elements, the center of the (0,0) element may be located arbitrarily within the unit cell (although centered elements are depicted in Figs. 2 and 3).

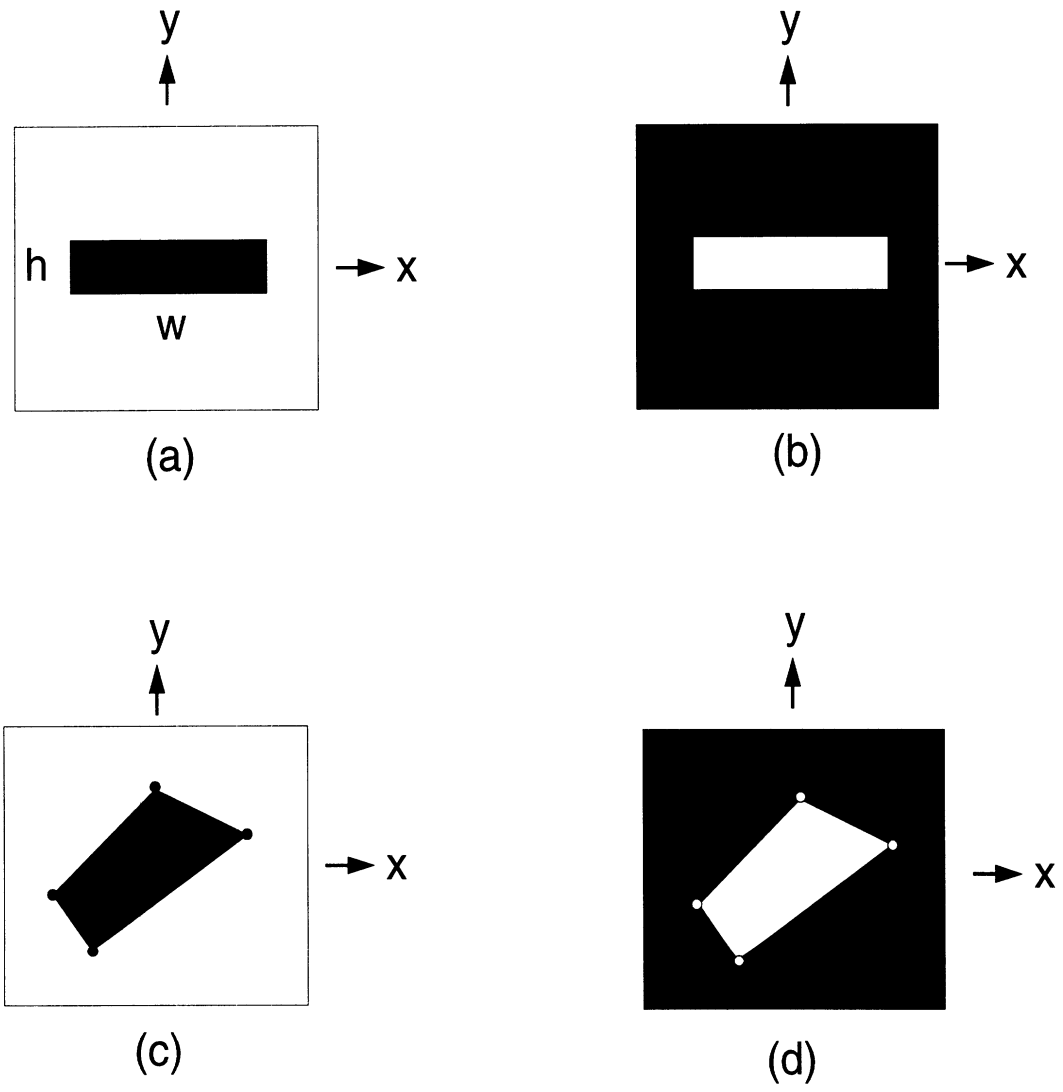

Figure 2. The rectangular patch element types from which the unit cell in a FSS array may be constructed, using the code FSSBUILD. (a) A metallic rectangular patch. (b) A rectangular slot in a ground plane. (c) A metallic quadrilateral patch. (d) A quadrilateral slot in a ground plane.

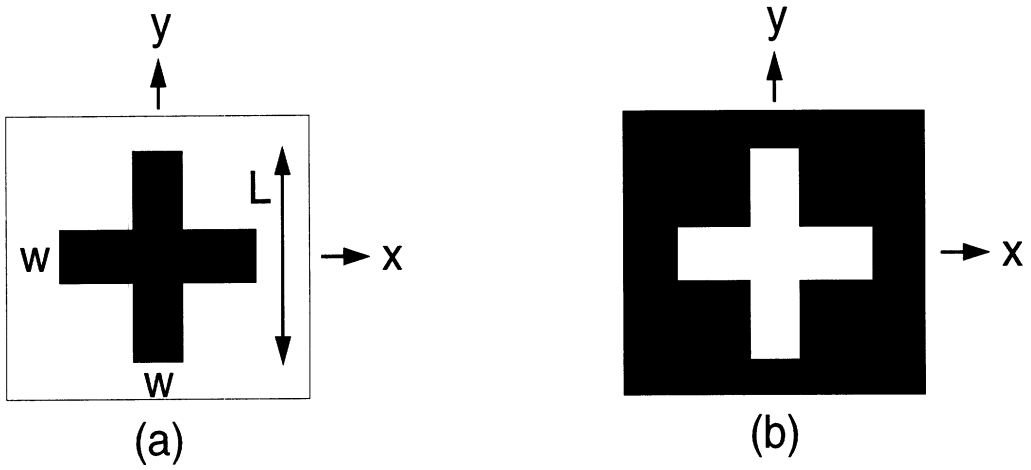

Figure 3. The cross elements that may be constructed using FSSBUILD. (a) The metallic cross. (b) The cross-shaped slot in a ground plane.

C. GRIDDING INFORMATION

The elements are gridded by FSSBUILD with triangular subdomains. Although FSSEIGER also allows the use of rectangular or even wire element subdomains, this capability has not been included in FSSBUILD at the present time. The user has control over the number of subdomains along the width and height of a rectangular element (Fig. 4 (a)), and independent control of the number of subdivisions along two adjacent edges of a general quadrilateral element (Fig. 4 (b)). An independent number of subdivisions can be specified along the length and width of one of the four identical arms for the cross elements (Fig. 4 (c)). The gridding of the center region of the cross is automatically determined once the number of subdivisions across the width of an arm is specified.

D. SOURCE AND LOAD INFORMATION

For radiation (antenna) problems, voltage sources may be placed across any of the edges in the mesh. For either radiation or scattering problems, lumped impedance loads may be placed across any of the edges in the mesh.

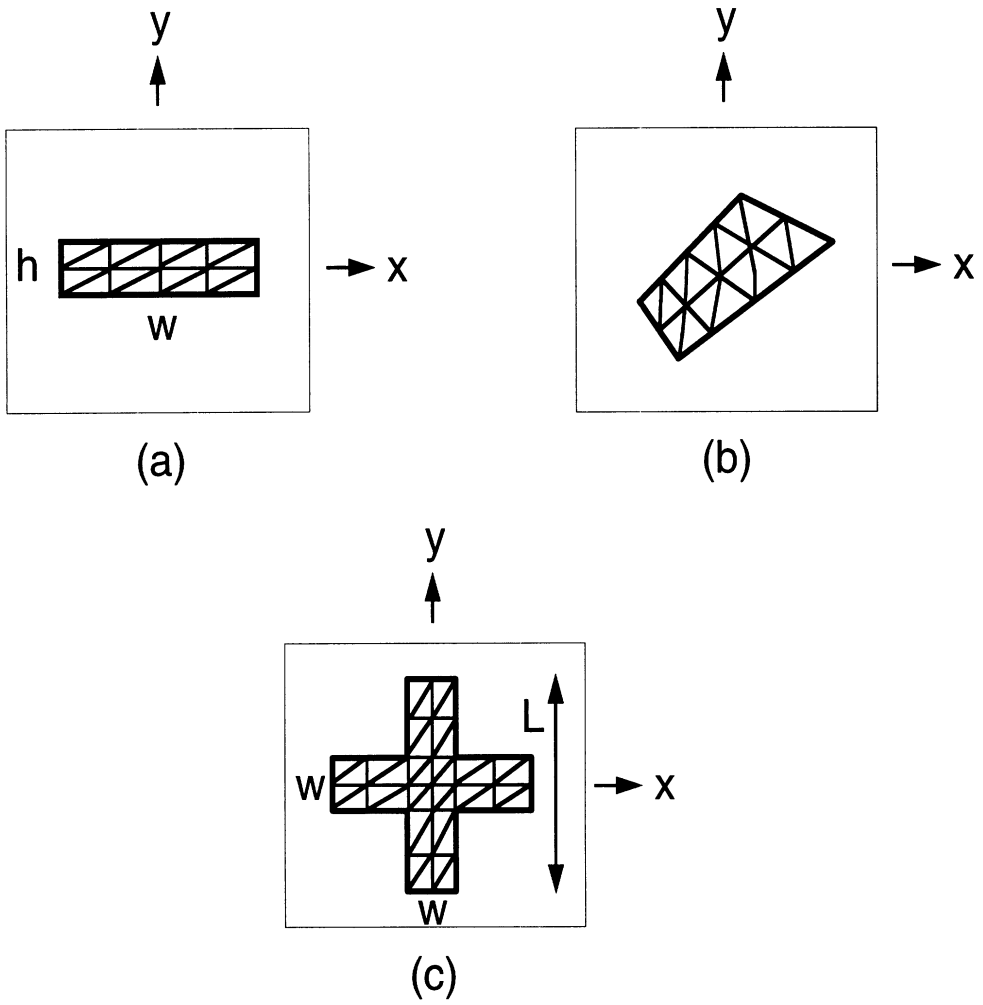

Figure 4. An illustration of the gridding scheme used by FSSBUILD, which subdivides the elements into triangular subdomains. (a) A rectangular element with 4 subdivisions along the width and 2 along the height. (b) A general planar quadrilateral with 4 subdivisions along one edge and 2 subdivisions along the adjacent edge. (c) A cross element with 2 subdivisions along the length and 2 subdivisions along the width of each arm.

E. LATTTICE INFORMATION

FSSBUILD allows the flexibility of specifying a unit cell that may be either rectangular or skewed. The geometries of the unit cell are shown in Figs. 5 (a) and (b) for the rectangular and skewed cases, respectively. For the rectangular lattice, the input parameters are the unit-cell dimensions in the x and y directions. For the skewed case, the user is asked to input the two lattice vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ that define the boundaries of the unit cell.

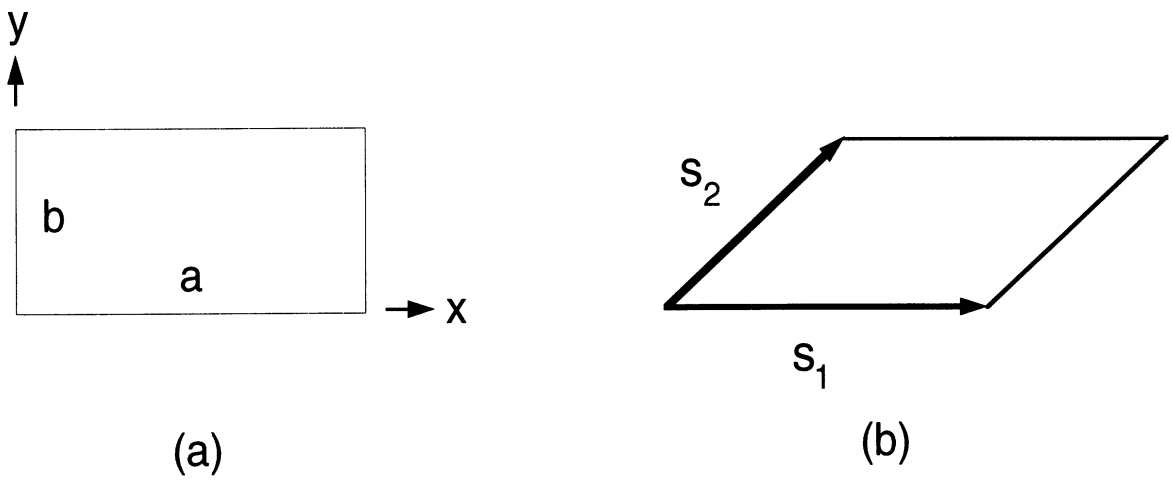

Figure 5. Geometry of the unit cell. (a) A rectangular unit cell is specified by the dimensions a and b along the x and y axes. (b) A skewed lattice is specified by the lattice vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ that define the boundaries of the unit cell.

III. RUNNING FSSBUILD

The FSSBUILD code is designed so that the user may enter the geometry data directly from the screen. Alternatively, a "script" file may be created, which contains the answers to the questions that the user would normally be interactively prompted for. Creating a script file is very helpful for the construction of large or complicated structures, or when slight modifications to an existing structure are desired. The interactive input of the data is described first, and then the use of a script file is discussed.

A. INTRODUCTION

The FSSBUILD code is designed to interactively prompt the user for the all of the necessary information required to generate the *.eig file, which is subsequently used as input to the FSSEIGER code when this code is run to solve the radiation or scattering problem. The interactive questions on the screen during the execution of FSSBUILD are designed to simplify the data entry as much as possible for the user, by making the specific questions dependent on the answers to preceding questions. For example, one question is whether or not all of the regions have a relative permeability of unity. If the answer is "yes", than the code will not query the user for relative permeabilities later. Similarly, if the user specifies that a rectangular patch element has all of the edges parallel to the x and y axes, the code will prompt the user for one of the rectangular elements shown in Fig. 2 (a) or (b), and not for the general quadrilateral element shown in Fig. 2 (c) or (d). The specific questions that are asked interactively are listed below.

B. SCREEN PROMPTS / EXPLANATIONS

FSSBUILD first asks the user a series of questions that determines the mesh geometry. Afterwards, the user answers questions to determine the layer geometry and the excitation. The geometry information part is complete after the user tells the code to save the data using the *save* command. The questions and explanations of the appropriate answers are explained here. For convenience, the questions are divided into separate sections, covering the geometry information, layer information, and excitation. All letters entered as input should be in lower case.

Mesh information

The following prompts are given to the user, and are used to determine the meshing information.

1. *Does geometry data exist:*
("y" = yes, "n" = no, "q" = quit)

The *yes* option means that an existing *.bld file already exists. The program will then ask for the name of the existing *.bld file. The *no* option means that a new *.bld file will be created. The *quit* option terminates the program.

2. *Enter one line problem description to be used as a file header.*

The description you input will be written to the top of the *.bld file, which then serves as a description of the file. For example, the description might be "two-layer FSS with TM incidence".

3. *Do you wish to...*
rect - add a rectangular element?
cros - add a cross element?
quer - query nodes, edges, and/or faces
modi - add, delete, or move a node or edge?
save - save geometry data and enter remaining code input?
rest - restart (start over)?
quit - quit and save nothing?

The first two options, *rect* and *cros*, will result in the code adding the indicated element type, which may either be a metallic or a slot element. If the z coordinate of the element coincides with the location of a ground plane, it will be assumed to be a slot element. Otherwise, it is assumed to be a metallic element. The *quer* option is used to obtain information about a node, edge, or face that already exists. The *modi* option is used to add, delete, or move a node or edge that already exists. The *save* option will prompt the user for the name of the *.bld file, and the existing element data will be written to that file. The *rest* option will restart the program over from the beginning. The *quit* option terminates the program without creating a *.bld file (unless

it has already been created previously using the save option). The first four options are described in more detail below.

If the "rect" option is chosen:

Enter z coordinate for planar rectangular structure:

The user is asked to input the z coordinate of the element, in meters. The z coordinate is arbitrary, and does not have to be at an interface between two regions. The origin ($z = 0$) is arbitrary. However, the user will be asked later to specify the z coordinates of the layer boundaries, and the origin should be consistent so that the elements are positioned properly within the layers.

Are the elements edges parallel to the x- and y- axes?
("y" = yes, "n" = no)

If the *yes* option is chosen, a rectangular element is assumed, with edges parallel to the x and y axes (Figs. 2 (a) and (b)). If the *no* option is chosen, a general planar quadrilateral is chosen (Figs. 2 (c) and (d)). The user will then be asked to input the coordinates of the four corners of the element.

For rectangular elements (parallel edges):

Enter center coordinates of rectangular element in the form "x,y":

If a rectangle with edges parallel to the x and y-axes was chosen, the user is now asked to input the coordinates of the center of the rectangle, with units in meters. For example, the center might be entered as "0.5, 0.75" (but without the quotes in the input expression).

Enter element width along x-axis in meters:

The width w along the x-dimension is entered in meters (see Fig. 2 (a) or (b)).

Enter no. of segments (subdomains) along the width:

The number of subdivisions along the width (x-dimension) is entered (see Fig. 4 (a)).

Enter height along y-axis in meters:

The height h along the y-dimension is entered in meters (see Fig. 2 (a) or (b)).

Enter no. of segments (subdomains) along the height:

The number of subdivisions along the height (y-dimension) is entered (see Fig. 4 (a)).

For general quadrilateral elements:

Enter x and y coordinates in the form "x,y" for the four corners in clockwise or counterclockwise sequence.

Corner 1?
Corner 2?
Corner 3?
Corner 4?

For each of the four corners, the coordinates are entered in meters. For example, corner 1 might entered as: "1.0, 2.5" (but without the quotes in the input expression).

How many edges (subdomains) along the edge formed by the first two corners?

Enter the number of subdivisions along the edge of the quadrilateral that runs between the first two corners that were input above. This will also be the number of subdivisions along the opposite edge (see Fig. 4 (b)).

How many edges along the adjacent side?

Enter the number of subdivisions along the edge of the quadrilateral that runs between corner 2 and corner 3. This will also be the number of subdivisions that runs along the opposite edge (see Fig. 4 (b)).

<u>If the "cros" option is chosen</u>:

Enter center coordinates of cross element in form "x,y,z":

The user inputs the coordinates of the center of the cross, in meters. For example, the center might be specified as "0.5, 0.5, 1.5" (but without the quotes in the input expression).

Enter the width of the cross arm in meters (all arms assumed identical):

The width *w* of the cross arms is entered in meters (all arms have the same width).

Enter no. of segments along the width of a cross arm:

Enter the number of subdivisions along the width of each arm of the cross (all arms are identically subdivided along their widths). The central portion of the cross to which all arms are attached is automatically subdivided based on the arm width segmentation information. (see Fig. 4 (c)).

Enter overall cross dimension (end of one arm to end of opposite arm):

Enter the overall dimension *L* (spanning two collinear arms of the cross) in meters (all arms are identical).

Enter no. of segments along length of one arm:

Enter the number of subdivisions along the length of an arm, i.e., from its point of attachment at the center of the cross to the tip of the arm. Each arm has the same number of subdivisions along its length. (see Fig. 4 (c)).

If the "quer" option is chosen:

Information on a node, edge, or face? Input "n" or "e" or "f" or "q"

Input the letter (without quotes) to find out information about either a node, edge, or face that has been previously created. The first three options are described below. The "*q*" option will abort this query.

Enter node number

Enter the number of the node you would like information on. For example, enter "1" (without the quotes) to determine information about node 1. The coordinates of the node are then displayed, as well as the edges that are connected to it.

Enter edge number

Enter the number of the edge you would like information on. For example, enter "1" (without the quotes) to determine information about edge 1. The program then responds by indicating which nodes are connected to the specified edge.

Enter face number

Enter the number of the face you would like information on. For example, enter "1" (without the quotes) to determine information about face 1. The program then responds by indicating which edges make up the face. Note: this option has not been implemented yet.

If the "modi" option is chosen:

Do you want to add, delete, or move? Input "a", "d", "m", or "q"

Input the letter (without quotes) to add, delete, or move a node or edge. The first three options are described below. The "*q*" option will abort this query.

Do you wish to add a node or insert edge? Input "n", "e", or "q".

Input the letter (without quotes) to insert a node or an edge. The "*q*" option aborts the insert process.

Input (x,y,z) for node newnode

Input the coordinates of the new node, in meters. The new node is automatically numbered with number newnode, which is one larger than the previously highest node number. The format for the entry is "*x,y,x*". Do not include the quotation marks or any parentheses in the input expression.

Input "from" and "to" nodes

Input the node numbers of the starting and ending nodes that define the new edge. These nodes must already exist. Note: No warning message is given if one of the nodes does not exist. The user must insure that the nodes already exist.

Delete nodes or edges? Input "n", "e", or "q"

Input the letter (without quotes) to indicate if it is a node or an edge that you wish to delete. The "*q*" option aborts the delete process.

Input range of node numbers
Input two node numbers, separated by a space. This range of node numbers will be deleted, and the remaining nodes will be automatically renumbered. For example, suppose the original structure has four nodes, numbered 1-4. If the range is input as "1 3", then the first three nodes will be deleted, and node 4 will be renumbered to become node 1.

Input edge number

Input the number of the edge you wish to delete. The remaining edges are renumbered, starting at number 1.

Which node?
Input the node number of the node you wish to modify. The program then prompts for the new (*x,y,z*) coordinates of that node. Enter the new coordinates in the format "*x,y,z*" (without the quotes).

Layer information

After the user has chosen the *save* option, the program provides the following prompt.

At this point you may begin to add layer and excitation data or quit and continue later. Do you wish to continue entering input data? (input "y" = yes or "n" = no)

The *yes* option allows the user to complete the data entry process for the layer and excitation data. The *yes* option will cause the code to prompt the user for the name of the *.eig file to which the meshing, layered media, and excitation information will be written. The *no* option allows the user to add this data later. Entering *no* will cause the program to terminate. Choosing the *yes* option results in the following prompt:

Count each of the following planar surfaces as an interface:
- a dielectric-to-dielectric interface
- a conducting ground plane (with or without slots)
Enter the number of layer interfaces:

At this point the user should enter the total number of interfaces, including the number of dielectric interfaces and the number of ground planes. (If a ground plane is at the boundary of

two different dielectric regions, this is counted as one interface, not two.) See Fig. 1 for the scheme for labeling interfaces.

The program then systematically prompts the user for the z coordinate for each of the interfaces, using the following prompt:

Layer boundary # k:
Enter the z-coordinate of the boundary:

Is boundary # k a ground plane?
(y = yes, n = no)

where k is the number of the interface. The user should first input the z coordinate in meters. The z coordinates of the layer boundaries relative to those of the elements (which have already been specified) determine the FSS locations within the layers. The answer to the second question determines whether the boundary layer is an interface between two different dielectric regions, or is a ground plane (possibly with slots). If the user specifies that the boundary layer is a ground plane, the program then asks if this ground plane contains a (periodic) slot element:

Does the conductor at boundary # k contain a slot element or aperture?
(y = yes, n = no)

If the answer is *yes*, the z coordinate of an element that was previously specified should agree with the z coordinate of this conducting interface. The z coordinates should agree to within a small number (the default value of which is 2.0E-05) in order for the program to treat the element as a slot element that is on the conducting interface. Otherwise, the element will be treated as a metallic element that is not on the conducting interface.

After the layer boundaries are established, the program will prompt the user for the complex relative permittivities and permeabilities. The program first asks the following question:

Is the relative permeability (mu_r) unity for all layers?
("y" = yes, "n" = no)

The user should input either "*y*" or "*n*" without the quotes. If the answer is "*y*", the program will not prompt for any permeabilities, thus simplifying data input. The program then prompts the user for the complex permittivity of each region, starting with region zero in Fig. 1, which is a half-space. The initial prompt is

Enter COMPLEX relative permittivity, (RE eps_r, IM eps_r),
for lower or left half space:

The complex relative permittivity for region zero should be entered, in the form indicated in the prompt, with the parentheses included. The program then systematically prompts the user for the complex relative permittivities of the different layers (1, 2,...,N -1 in Fig. 1). The number of

layers is determined by the number of layer interfaces that was previously entered. The form of the prompt is:

Layer # k:
Enter COMPLEX relative permittivity, (RE eps_r, IM eps_r):

The complex relative permittivity is entered in the same format as mentioned above (including the parentheses). After the data for layer N has been specified, the program prompts the user for the complex permittivity of the upper half space, region N in Fig. 1. The prompt is

Enter COMPLEX relative permittivity, (RE eps_r, IM eps_r),
for upper or right half space:

This value should be entered using the same format as for the other regions.

Lattice information

After the layer permittivities have been entered (the last one being region N), the program prompts the user for information about the lattice dimensions. The element locations specified previously for each FSS array are actually the coordinates for the center of the (0,0) unit cell, and the periodicity of the FSS lattices determines the layout of the other elements in that FSS. All FSS layers are assumed to have the same lattice periodicity in this version of the code. The program first asks the user if the lattice is rectangular with the prompt

Is the periodic lattice rectangular?
("y" = yes, "n" = no)

If the answer is "*y*", the program asks only for the dimensions of the unit cell. If the answer is "*n*", the program asks for two lattice vectors that determine the boundaries of the unit cell (see Fig. 5).

For a rectangular lattice the next prompt is as follows:

Enter x and y lattice dimensions, respectively:

The x and y dimensions, labeled as a and b in Fig. 5 (a), are entered in meters. The format is "x y". that is, the dimension x is entered, then a space, and then the y dimension. Do not include the quotes in the input expression.

For a skewed lattice the next prompt is as follows:

Enter lattice vectors s_1 and s_2 defining the boundaries of a unit cell of the periodic structure.
(The cross product s_1 × s_2 should point in the direction of the + z-axis.)

Enter s_1 vector as "s_1_x, s_1_y" (s_1z assumed 0):
Enter s_2 vector as "s_2_x, s_2_y" (s_2z assumed 0):

The x and y components of the lattice vectors s_1 and s_2 are entered as indicated, with dimensions in meters.

The program then gives the following message:

*All geometrical data has now been entered and only excitation data remains. You may wish to stop and examine the *.eig file now to locate edges at which voltage sources are applied.*

Do you wish to enter remaining input?
(input "y" or "n")

A "*y*" response will allow the user to continue entering the excitation data. A "*n*" response will terminate the program. The user may wish to terminate the program now if the problem is a radiation (antenna) problem, because the user will want to know at which edges to put voltage sources. This can be done by examining the *.eig file to determine the edge numbers. (The edge numbers could have also been determined by using the *quer* option interactively, as described previously). For a scattering problem, there is no need to examine the *.eig file at this point unless one or more elements has a lumped load across an edge.

Lumped load information

Assuming a "*y*" response above, the next information the user is prompted for is the number of lumped loads. If this number is greater than zero, the program systematically prompts the user for the value and location of each complex load impedance. The prompt has the form

Enter complex impedance in Ohms, the element number to which one terminal is attached, the local node number opposite the edge to which it is attached.

(Re Z, Im Z), element number, local node number

The user first inputs the complex load impedance in the format indicated in the prompt, including the parentheses and comma. After the load impedance is entered, the number of the element (triangle) that contains the edge at which the load is to be connected is entered. Finally, the local node number of the node on element *element number* that is opposite to the edge containing the load is entered. The load is then assumed to be connected between the two triangles that share this edge. The triangle number and the local node number can be obtained by examining the *.eig file.

Excitation information

After the load impedances are entered, the excitation data is entered. The excitation may consist of voltage sources across edges for a radiation (antenna) problem, or an incident plane wave, for a scattering problem.

Scan or incidence angles:

The first prompt is for the angles of incidence of an incident plane wave, or the scan angles for a phased array:

Enter the scan or incidence angle ranges in degrees.
Beginning theta, ending theta, number of theta values:

The angle θ is the usual spherical coordinate angle θ measured from the z-axis. It is interpreted as the array scan angle for a radiation (antenna) problem and as the angle of an incident ray for a scattering problem. In either case, the angle is in degrees. The user inputs the beginning scan (or incidence) angle, followed by the ending scan (or incidence) angle, and then the number of angles to be calculated between the beginning and ending angles. For example, the input "30.0, 60.0, 31" would produce output at every degree, starting at 30 degrees and ending at 60 degrees.

The user is then prompted for the angle range in ϕ over which the scan or incidence angle ranges. The angle ϕ is measured from the x-axis and is the projection of the scan or incidence direction onto the x-y plane. The prompt is

Beginning phi, ending phi, number of phi values

This data is input with the same format as for θ, described above. The angle is in degrees.

Voltage source information:

The program then queries the user for any voltage sources that might be present (indicating that the problem is a radiation or antenna problem). The program checks this by issuing the following prompt:

Multiple (simultaneous) voltage sources are allowed.
How many voltage source locations
exist for each excitation?

The user inputs the number of voltage sources. The number of voltage sources is assumed to be the same for all scan angles. If there is at least one voltage source, the program assumes that this is a radiation (antenna) problem. If there are no voltage sources (the number zero is entered) the program assumes that this is a scattering problem, and will prompt the user for information about the incident plane wave. The two cases are discussed separately.

The program asks the user to specify the complex voltage of the source, and to identify the edge at which the source is connected. The voltage source is assumed to maintain a specified voltage between two elements having the edge in common. The following prompt is given systematically for each source:

Enter voltage in COMPLEX format, the element number of positive reference terminal, local node number opposite edge to which voltage source is attached.
(Re V, Im V), element number, local_node

The complex voltage is first entered in the format shown, including the parentheses and comma. This is followed by the number of the triangle element to which the positive reference terminal of the voltage source is attached. This is followed by the local node number of the node on element *element number* opposite the edge at which the source is attached. The information regarding the triangle number and local node number can be obtained by examining the *.eig file.

Incident plane wave information:

The program prompts the user to provide the properties of the incident plane wave. The first prompt is:

Incident plane waves are normalized to have unit electric field components in the plane transverse to the z-axis. They may be polarized TE or TM to this plane. Only the polarization needs to be specified.

Is the polarization of the incident plane wave TE or TM to the z-plane?
("TE" or "TM")

The phrase "TE" or "TM" (without the quotes) is entered to specify the polarization of the incident plane wave.

Frequency data

After the specification of the voltage sources or the incident plane wave properties, the user is requested to enter the frequencies at which the calculations are to be performed. The prompt is

Enter beginning and ending frequencies (in Hz) and
the number of excitation frequencies:

The user inputs this data with a space between entries. For example, the input "10.0E+9 15.0E+09 6" would result in a calculation at every 1.0 GHz from 10.0 to 15.0 GHz.

Computational parameters

The program prompts the user to enter data for the limits on the spectral sums used to compute the layered media Green's functions. The settings of these sum limits are a tradeoff between accuracy and computation time, and acceptable values usually must be determined by a trial and error process. Generally speaking, higher sum limits are needed when array or FSS elements are near or at an interface, especially if the interface involves a high dielectric contrast. Also if the elements are near thin layers, *M_limit* and *N_limit* may need to be set to higher values to achieve convergence. For periodic structures whose unit cells have a high aspect ratio, the limits should be set such that the relation $|\mathbf{s}_1|\, N\_limit = |\mathbf{s}_2|\, M\_limit$ is approximately true. The prompt to enter the summation limits is

Enter summation limits, M_limit, N_limit for spectral sums:

The user inputs two integers corresponding to *M_limit* and *N_limit,* separated by a space.

Next, the user is prompted for the number of quadrature points used in the testing scheme on the triangles. Usually this number can be taken as 1 for maximum efficiency; for some applications, however, greater accuracy is obtained by choosing a higher number. The prompt is

Enter number of quadrature points for testing on triangles.
Must be 1,3, or 7:

An integer having the value 1,3, or 7 may be entered; any other value will cause an error.

This completes the data entry for FSSBUILD.

C. SCRIPT FILES

As an alternative to entering the data interactively from the screen, a "script" file may be created and read in as input. This file is an ASCII file that contains the answers to the same questions that would be asked interactively. Since the answer to certain interactive questions determines what the following questions will be, it is very important that the user pay close attention to the entries in the script file. A sample script file is given in Appendix B. A discussion of how the script file is read into the FSSBUILD program using Digital Visual Fortran is included in the discussion given in Appendix A.

D. OUTPUT FROM FSSBUILD

The FSSBUILD code produces two output files, the *.eig file and the *.bld file. The *.bld file may be considered a temporary file containing descriptions of the nodes and edges. The *.eig file is used as the input to FSSEIGER. A sample *.eig file, also corresponding to the *.scp file in Appendix B, is listed in Appendix C.

IV. OUTPUT FROM FSSEIGER

Once the *.eig file is obtained from FSSBUILD, the set of codes called FSSEIGER may be run. When FSSEIGER is executed it will ask for the name of the *.eig file to be used as input; since the code assumes the .eig extension, the extension should not be entered by the user. The code will then ask for the name of the output file to which the results will be written; again, the file extension should not be entered, as the code assumes it to be *.mnh. The code will also prompt for the name of a file to which any error messages will be written; the code assumes the extension for this file to be *.err.

A sample *.mnh output file, corresponding to the *.scp file in Appendix B, is included in Appendix D. In addition to listing some of the user-specified excitation data for each case, the *.mnh file lists the condition number of the moment-method matrix solved so the user can monitor any potential problems due to ill conditioning. The output file also contains the electric and / or magnetic surface current densities normal to the interior edges of the mesh. For radiation (antenna) problems with a single voltage source, the so-called active input impedance and admittance are listed. The input power per element is also given. Also listed are the components of the electric far field vector in the direction of the array scan. This far-field is obtained by taking the calculated currents on a particular element (the (0,0) element) in the *periodically-excited* phased array, and calculating the far-field from these same currents on a *nonperiodic* element (i.e., all other elements are assumed to be absent). It can be proven that the resulting radiation field, calculated in the direction of the array scan, is equal to the *element pattern of the phased array* in that direction. The element pattern of a phased array is the pattern that would be obtained if a single element in the array were excited (with the same voltage sources that were present on the (0,0) element of the phased array), with all other elements *present but passive* (voltage sources short-circuited on those elements). Since the code only outputs the far-field in the direction of the array scan, it would be necessary to rerun the code with different scan angles in order to obtain the complete element pattern of the phased array for all observation angles.

For a scattering problem, the *.mnh file lists the reflection and transmission coefficients of the periodic structure for both the TE and TM polarizations. The reflected waves are those

propagating at the specular reflection angle, designated as S11_00_00_TE and S11_00_00_TM, respectively. S11 indicates that the excitation is from side "1" of the array and that the scattered fields on side "1" are observed. The 00_00 notation indicates the (0,0) Floquet mode reflected from the structure due to an incident (0,0) Floquet mode (either TE or TM). The transmission coefficients are designated S21_00_00_TE and S21_00_00_TM, where S21 denotes the field transmitted through the array to side "2" due to an incident field on side "1." Unfortunately, the current version of FSSBUILD does not set up data for FSSEIGER so that one may obtain S21_00_00_TE and S21_00_00_TM for both polarizations simultaneously. Hence if one wishes to obtain data for excitation by the opposite polarization—even for the same angle of incidence—the problem must be re-solved. To obtain S22_00_00_TX and S12_00_00_TX, (X=E or M), one must merely change the angle of incidence θ such that the excitation is from side "2" of the array, re-solve the problem, and interchange the interpretation of "1" and "2" in the results.

The output file also lists the squares of the magnitudes of the reflection and transmission coefficients and their values in decibels. A "power check" is then given, which gives the sums of the squares of the magnitudes of the TE and TM waves for both the reflected and transmitted fields. For lossless structures with no higher-order Floquet modes propagating, these quantities, |S11_00_00_TE|**2 + |S11_00_00_TM|**2 and |S21_00_00_TE|**2 + |S21_00_00_TM|**2, should sum to unity since power must be conserved. Hence, for lossless structures, this sum can serve as a check on the validity of the numerical results. Failure to sum to unity, on the other hand, suggests perhaps that the parameters *M_limit, N_limit*, and/or the number of testing points should be increased. For lossless structures, the sum of the squared magnitudes of the reflection and transmission coefficients in principle approaches unity as these numerical parameters tend to infinity, *regardless of the geometrical subdivision scheme*. Hence summing to unity is *not* a sufficient check on the convergence of the geometrical subdivision scheme.

APPENDIX A
DIGITAL VISUAL FORTRAN 90 SETUP

I. Recommended directory structure

A recommended directory structure for the FSSBUILD and FSSEIGER codes, and the various input and output files, is shown below. We describe the setup as it might appear on a workstation using Digital Visual Fortran 90.

FSSSUITE (top directory)

FSSBUILD (FSSBUILD codes)

Main.f90

FSSbuild.f90

Writebld.f90

Cross.f90

Filemod.f90

Iounit_m.f90

FSSEIGER (FSSEIGER codes)

There are a total of 43 *.f and *.f90 modules in this directory. The *.f files are fixed format source files, while the *.f90 files use free format.

asmbly_m.f90	basis_m.f90	cerfc.f	cmptpt_m.f90
cnstnt_m.f90	curvpt.f	eiger.f90	elemat_m.f90
elemnt_m.f90	elevec_m.f90	excit_m.f90	filemod.f90
FSSEiger_m.f90	gauss_m.f90	geom_m.f90	gltramod.f90
initr.f	intrfc_m.f90	iounit_m.f90	juncpt_m.f90
linear_m.f90	load_m.f90	matrix_m.f90	numparm.f90
pproj.f	p_file.f90	p_period.f90	p_potent.f90
p_somint.f90	p_spfun.f90	p_tline.f90	reg_m.f90

solutn_m.f90	solver.f	stript.f	surfpt_m.f90
sylpy_m.f90	triapt.f	tripar.f	tubept.f90
utilmod.f90	vecasy_m.f90	vector_m.f90	

SCPFILES (directory for storing *.scp files)

BLDFILES (directory for storing *.bld files)

EIGFILES (directory for storing *.eig files)

OUTFILES (directory for storing *.mnh files)

TEMPEIG (directory for storing *.eig files while testing installed code)

TEMPBLD(directory for storing *.bld files while testing installed code)

II. Digital Visual Fortran setup notes

To compile, link, and execute the Fortran programs in the FSSBUILD or FSSEIGER directories, follow these steps:

a) From the "File" menu, click on New, and select Projects.

b) Click on "Win32Console Application", and enter the name of the project workspace. Then chose the project "location" (the browser button may be used to locate the project in the FSSBUILD or FSSEIGER folders with object and executable files in subdirectories just below them).

c) From the "Project" menu, choose "Add to Project" and then "Files". To build FSSBUILD or FSSEIGER, select all of the *.f or *.f90 files in the FSSBUILD or FSSEIGER directories.

d) In the Project / Settings / Fortran / Fortran Data menu, set Default Real to 8. This will maintain the correct level of precision ("double precision") required by the codes.

e) To compile and link the Fortran files in the project, select the "Build" menu and then select "Update all Dependencies."

f) Select "Build FSSBUILD.exe" (or FSSEIGER.exe) to run the compiled and linked program.

g) To read input from a script file rather than from the console, go to the Project / Settings / Debug menu , and enter in the "Program Arguments" box the path and name to the *.scp file in the format <path\filename.scp name. For example, the entry might be "<d:\FSSsuite\scpfiles\pozarvolt.scp"

APPENDIX B

SAMPLE SCRIPT FILE

The following script file, pozarvolt.scp, is designed to solve for the input impedance of an infinite phased array of printed dipoles on a single grounded dielectric layer. The file has been commented so that the input lines can be easily understood.

```
n                    ! data does not exist; title on next line
Pozar dipole, voltage excited, unloaded, single scan angle & freq.
rect                 ! build a rectangle
0.0                  ! z-coordinate of rectangle
y                    ! yes, rectangles is aligned with x-,y-axes
0. 0.                ! x,y coordinates of center of rectangle
0.01                 ! width of rectangle along x-axis
1                    ! number of subdivisions along x-axis
0.39                 ! height of rectangle along y-axis
12                   ! number of subdivisions along y-axis
save                 ! save geometry in a *.bld file
d:\FSSsuite\bldfiles\pozarvol          ! name of *.bld file
y                    ! yes, continue entering data
d:\FSSsuite\eigfiles\pozarvol          ! name of *.eig file
2                    ! number of interfaces
-0.19                ! z-coordinate of 1st interface
y                     ! yes, it is a ground plane
```

```
n                  ! no, it doesn't contain a slot
0.0                ! z-coordinate of 2nd interface
n                  ! no, it isn't a ground plane
y                  ! permeabilities are all unity
(2.55,0.0)         ! complex relative epsilon of layer # 1
(1.0,0.0)          ! complex relative epsilon of upper half space
y                  ! lattice is rectangular
0.5  0.5           ! x, y dimensions of rectangular lattice
y                  ! yes, enter remaining input
0                  ! number of lumped loads
1.0  1.0  1        ! theta_start, theta_end, # theta values
0.0  0.0  1        ! phi_start, phi_end, # phi values
1                  ! # voltage source excitations
(1.0,0.0)  13  1   ! complex voltage; element, local node excited
3.0e8   3.0e8  1   ! begin freq, end freq, and # of frequencies
1  1               ! sum limits on periodic Green's function
1                  ! number of testing points on each triangle
```

APPENDIX C

SAMPLE EIG FILE

This sample*.eig file, pozarvolt.eig, was obtained from running FSSBUILD using the script file pozarvolt.scp.

---

pozar dipole, voltage excited, unloaded, single scan angle & freq

```
1
1  pec
1
1 1 3 triangle
        5
        1 ordinary vector T F pec_efie 1 3
         1
           1 1          0
        2 ordinary vector T F pec_efie 1 3
         1
           1 1          1
        3 ordinary vector T F pec_efie 1 3
         1
           2 1          1
        4 ordinary vector T F pec_efie 1 3
         1
           2 1          2
        5 ordinary vector  F  T  aperture_gndp 1 3
         2
           1        2       -1        1        1        1
```

```
      2
      1 layered periodic         2
            (  1.00000,  0.00000)   (  1.00000,  0.00000)
  -0.190000E+00 (  1.00000,  0.00000)   (  1.00000,  0.00000)
  -0.190000E+00 (  1.00000,  0.00000)   (  1.00000,  0.00000)
      0        1
0.500000000000000     0.000000000000000E+000     0.0
0.000000000000000E+000 0.500000000000000         0.0
      2 layered periodic         2
            (  1.00000,  0.00000)   (  1.00000,  0.00000)
  -0.190000E+00 (  2.55000,  0.00000)   (  1.00000,  0.00000)
   0.000000E+00 (  1.00000,  0.00000)   (  1.00000,  0.00000)
      1        0
0.500000000000000     0.000000000000000E+000     0.0
0.000000000000000E+000 0.500000000000000         0.0
 26  24  23
  1       -5.000000E-03       -1.950000E-01        0.000000E+00
  2        5.000000E-03       -1.950000E-01        0.000000E+00
  3       -5.000000E-03       -1.625000E-01        0.000000E+00
  4        5.000000E-03       -1.625000E-01        0.000000E+00
  5       -5.000000E-03       -1.300000E-01        0.000000E+00
  6        5.000000E-03       -1.300000E-01        0.000000E+00
  7       -5.000000E-03       -9.750000E-02        0.000000E+00
  8        5.000000E-03       -9.750000E-02        0.000000E+00
  9       -5.000000E-03       -6.500000E-02        0.000000E+00
 10        5.000000E-03       -6.500000E-02        0.000000E+00
 11       -5.000000E-03       -3.250000E-02        0.000000E+00
 12        5.000000E-03       -3.250000E-02        0.000000E+00
 13       -5.000000E-03        0.000000E+00        0.000000E+00
 14        5.000000E-03        0.000000E+00        0.000000E+00
 15       -5.000000E-03        3.250000E-02        0.000000E+00
 16        5.000000E-03        3.250000E-02        0.000000E+00
```

```
17      -5.000000E-03      6.500000E-02      0.000000E+00
18       5.000000E-03      6.500000E-02      0.000000E+00
19      -5.000000E-03      9.750000E-02      0.000000E+00
20       5.000000E-03      9.750000E-02      0.000000E+00
21      -5.000000E-03      1.300000E-01      0.000000E+00
22       5.000000E-03      1.300000E-01      0.000000E+00
23      -5.000000E-03      1.625000E-01      0.000000E+00
24       5.000000E-03      1.625000E-01      0.000000E+00
25      -5.000000E-03      1.950000E-01      0.000000E+00
26       5.000000E-03      1.950000E-01      0.000000E+00
     1  triangle  1  1  1
     3
 4   2   1
 0   1   0
 1
     2  triangle  1  1  1
     3
 3   4   1
 1   0   1
-1
 2
     3  triangle  1  1  1
     3
 6   4   3
 1   1   0
-2
 3
     4  triangle  1  1  1
     3
 5   6   3
 1   0   1
-3
```

```
4
     5  triangle  1  1  1
     3
8   6   5
1   1   0
-4
5
     6  triangle  1  1  1
     3
7   8   5
1   0   1
-5
6
     7  triangle  1  1  1
     3
10   8   7
1   1   0
-6
7
     8  triangle  1  1  1
     3
9   10   7
1   0   1
-7
8
     9  triangle  1  1  1
     3
12  10   9
1   1   0
-8
9
     10  triangle  1  1  1
```

```
      3
11   12    9
 1    0    1
-9
10
      11   triangle   1  1  1
      3
14   12   11
 1    1    0
-10
11
      12   triangle   1  1  1
      3
13   14   11
 1    0    1
-11
12
      13   triangle   1  1  1
      3
16   14   13
 1    1    0
-12
13
      14   triangle   1  1  1
      3
15   16   13
 1    0    1
-13
14
      15   triangle   1  1  1
      3
18   16   15
```

```
 1   1   0
-14
 15
      16  triangle  1  1  1
      3
 17  18  15
 1   0   1
-15
 16
      17  triangle  1  1  1
      3
 20  18  17
 1   1   0
-16
 17
      18  triangle  1  1  1
      3
 19  20  17
 1   0   1
-17
 18
      19  triangle  1  1  1
      3
 22  20  19
 1   1   0
-18
 19
      20  triangle  1  1  1
      3
 21  22  19
 1   0   1
-19
```

```
 20
      21   triangle   1  1  1
       3
 24   22   21
  1    1    0
-20
 21
      22   triangle   1  1  1
       3
 23   24   21
  1    0    1
-21
 22
      23   triangle   1  1  1
       3
 26   24   23
  1    1    0
-22
 23
      24   triangle   1  1  1
       3
 25   26   23
  1    0    0
-23
       0
       1
       0          1
(1.00000000000000,0.000000000000000E+000)         13          3          1
periodic
  1.00000000000000       0.000000000000000E+000
       1
  300000000.000000       1
```

1 1

1

APPENDIX D

SAMPLE OUTPUT FILE

This sample *.mnh file, pozarvolt.mnh, was obtained from running the EIGER input file pozarvolt.eig.

---

pozar dipole, voltage excited, unloaded, single scan angle & freq

==========

Condition number is 1088.14845788561

Excitation Data:

Voltage source excitation:
Number of voltage_sources: 1

Voltage source # 1 :
(1.00000000000000,0.000000000000000E+000) volt source
has positive reference location opposite local node 1
on element number 13

Frequency = 300000000.000000 (Hz)

Scan or incidence angles:
theta = 1.00000000000000 (degrees)
phi = 0.000000000000000E+000 (degrees)

The following list of currents may contain electric and/or
magnetic currents.

Currents

1 (6.565731934023700E-002,-8.674246712370985E-003)

2 (0.400795342207085,-4.630691065875420E-002)

3 (0.165804360963758,-1.631057668891133E-002)

4 (0.715116278438726,-5.841633214872675E-002)

5 (0.250702495245538,-1.606402344797198E-002)

6 (0.976468970289026,-4.505386186626879E-002)

7 (0.318688285073875,-8.531682677355471E-003)

8 (1.17468548004381,-7.282339940805895E-003)

9 (0.366413281852113,6.217523756258755E-003)

10 (1.29874304892746,5.595956688213395E-002)

11 (0.391052198916468,2.932048279695615E-002)

12 (1.34099525963193,0.172100365619750)

13 (0.391052198903325,2.932048401338074E-002)

14 (1.29874304928390,5.595956577662674E-002)

15 (0.366413281810504,6.217527209712278E-003)

16 (1.17468548067320,-7.282341933284076E-003)

17 (0.318688284997910,-8.531677541812327E-003)

18 (0.976468971037168,-4.505386434331438E-002)

19 (0.250702495127010,-1.606401745189373E-002)

20 (0.715116279103085,-5.841633458170881E-002)

21 (0.165804360792612,-1.631057077953985E-002)

22 (0.400795342540654,-4.630691242167102E-002)

23 (6.565731911993074E-002,-8.674241790833849E-003)

Active input admittance and impedance:

admittance = (1.340995259631933E-002,1.721003656197502E-003) Mhos

impedance = (73.3631449903800,-9.41526376411248) Ohms

Input power = 6.704976298159667E-003 Watts/element

Normalized far field element pattern, front of array

E_phi*kr*e^(jkr) = (-3.48063892696124,-6.15374612713675)

E_theta*kr*e^(jkr) = (8.145507192787088E-003,5.328939165829523E-003)

Normalized far field element pattern, rear of array

E_phi*kr*e^(jkr) = (0.000000000000000E+000,0.000000000000000E+000)

E_theta*kr*e^(jkr) = (0.000000000000000E+000,0.000000000000000E+000)

APPENDIX E

SAMPLE MAKE FILE

A sample MAKE file for compiling FSSEIGER in a UNIX environment follows.

```
EIGER_DEP = filemod.o gltramod.o utilmod.o iounit_m.o cnstnt_m.o \
        vector_m.o sylpy_m.o numparm.o gauss_m.o linear_m.o reg_m.o \
        elemnt_m.o geom_m.o basis_m.o tubept.o surfpt_m.o juncpt_m.o \
        intrfc_m.o excit_m.o p_file.o p_spfun.o p_tline.o p_period.o \
        p_somint.o p_potent.o cmptpt_m.o elemat_m.o asmbly_m.o elevec_m.o \
        vecasy_m.o matrix_m.o load_m.o fsseiger_m.o solutn_m.o eiger.o curvpt.o \
        initr.o solver.o pproj.o stript.o triapt.o tripar.o cerfc.o
#         fmin.o lnsrch.o lubksb.o ludcmp.o newt.o

EIGER_DB_DEP = filemod.do gltramod.do utilmod.do iounit_m.do cnstnt_m.do \
          vector_m.do sylpy_m.do numparm.do gauss_m.do linear_m.do reg_m.do \
          elemnt_m.do geom_m.do basis_m.do tubept.do surfpt_m.do \
          juncpt_m.do intrfc_m.do excit_m.do p_file.do p_spfun.do \
          p_tline.do p_period.do p_somint.do p_potent.do cmptpt_m.do \
          elemat_m.do asmbly_m.do elevec_m.do vecasy_m.do matrix_m.do \
          load_m.do fsseiger_m.do solutn_m.do eiger.do curvpt.do initr.do solver.do \
          pproj.do stript.do triapt.do tripar.do cerfc.do
#           fmin.do lnsrch.do lubksb.do ludcmp.do newt.do

EIGER_FLAGS = -c -r8 -O

EIGER_DB_FLAGS = -c -r8 -g -check underflow -check overflow -check bounds -ladebug

eiger: $(EIGER_DEP)
        f90 -o eiger $(EIGER_DEP)
```

```
fsseiger_m.o: fsseiger_m.f90 matrix_m.o iounit_m.o vector_m.o geom_m.o \
        excit_m.o reg_m.o elevec_m.o cnstnt_m.o vecasy_m.o
        f90 $(EIGER_FLAGS) -o fsseiger_m.o fsseiger_m.f90
numparm.o: numparm.f90 reg_m.o iounit_m.o
        f90 $(EIGER_FLAGS) -o numparm.o numparm.f90
filemod.o: filemod.f90
        f90 $(EIGER_FLAGS) -o filemod.o filemod.f90
gltramod.o: gltramod.f90
        f90 $(EIGER_FLAGS) -o gltramod.o gltramod.f90
utilmod.o: utilmod.f90
        f90 $(EIGER_FLAGS) -o utilmod.o utilmod.f90
iounit_m.o: iounit_m.f90
        f90 $(EIGER_FLAGS) -o iounit_m.o iounit_m.f90
cnstnt_m.o: cnstnt_m.f90
        f90 $(EIGER_FLAGS) -o cnstnt_m.o cnstnt_m.f90
vector_m.o: vector_m.f90
        f90 $(EIGER_FLAGS) -o vector_m.o vector_m.f90
sylpy_m.o: sylpy_m.f90 vector_m.o
        f90 $(EIGER_FLAGS) -o sylpy_m.o sylpy_m.f90
gauss_m.o: gauss_m.f90 cnstnt_m.o iounit_m.o numparm.o
        f90 $(EIGER_FLAGS) -o gauss_m.o gauss_m.f90
linear_m.o: linear_m.f90 vector_m.o iounit_m.o
        f90 $(EIGER_FLAGS) -o linear_m.o linear_m.f90
reg_m.o: reg_m.f90 vector_m.o cnstnt_m.o gltramod.o iounit_m.o
        f90 $(EIGER_FLAGS) -o reg_m.o reg_m.f90
elemnt_m.o: elemnt_m.f90 reg_m.o linear_m.o sylpy_m.o vector_m.o cnstnt_m.o \
        iounit_m.o
        f90 $(EIGER_FLAGS) -o elemnt_m.o elemnt_m.f90
geom_m.o: geom_m.f90 elemnt_m.o iounit_m.o
        f90 $(EIGER_FLAGS) -o geom_m.o geom_m.f90
basis_m.o: basis_m.f90 elemnt_m.o cnstnt_m.o
        f90 $(EIGER_FLAGS) -o basis_m.o basis_m.f90
```

```
tubept.o: tubept.f90 vector_m.o
	f90 $(EIGER_FLAGS) -o tubept.o tubept.f90
surfpt_m.o: surfpt_m.f90 elemnt_m.o gauss_m.o vector_m.o cnstnt_m.o iounit_m.o
	f90 $(EIGER_FLAGS) -o surfpt_m.o surfpt_m.f90
juncpt_m.o: juncpt_m.f90 surfpt_m.o elemnt_m.o gauss_m.o vector_m.o \
	cnstnt_m.o iounit_m.o
	f90 $(EIGER_FLAGS) -o juncpt_m.o juncpt_m.f90
intrfc_m.o: intrfc_m.f90 surfpt_m.o tubept.o elemnt_m.o vector_m.o iounit_m.o
	f90 $(EIGER_FLAGS) -o intrfc_m.o intrfc_m.f90
excit_m.o: excit_m.f90 reg_m.o vector_m.o cnstnt_m.o iounit_m.o utilmod.o
	f90 $(EIGER_FLAGS) -o excit_m.o excit_m.f90
p_file.o: p_file.f90 reg_m.o cnstnt_m.o
	f90 $(EIGER_FLAGS) -o p_file.o p_file.f90
p_spfun.o: p_spfun.f90
	f90 $(EIGER_FLAGS) -o p_spfun.o p_spfun.f90
p_tline.o: p_tline.f90 p_file.o p_spfun.o
	f90 $(EIGER_FLAGS) -o p_tline.o p_tline.f90
p_period.o: p_period.f90 elemnt_m.o vector_m.o numparm.o
	f90 $(EIGER_FLAGS) -o p_period.o p_period.f90
p_somint.o: p_somint.f90 p_tline.o p_spfun.o p_file.o vector_m.o iounit_m.o
	f90 $(EIGER_FLAGS) -o p_somint.o p_somint.f90
p_potent.o: p_potent.f90 p_somint.o p_period.o p_tline.o p_spfun.o \
	p_file.o excit_m.o juncpt_m.o intrfc_m.o elemnt_m.o \
	linear_m.o gauss_m.o vector_m.o
	f90 $(EIGER_FLAGS) -o p_potent.o p_potent.f90
cmptpt_m.o: cmptpt_m.f90 p_potent.o p_file.o juncpt_m.o intrfc_m.o elemnt_m.o \
	gauss_m.o vector_m.o cnstnt_m.o iounit_m.o
	f90 $(EIGER_FLAGS) -o cmptpt_m.o cmptpt_m.f90
elemat_m.o: elemat_m.f90 cmptpt_m.o basis_m.o elemnt_m.o gauss_m.o vector_m.o \
	cnstnt_m.o iounit_m.o
	f90 $(EIGER_FLAGS) -o elemat_m.o elemat_m.f90
asmbly_m.o: asmbly_m.f90 elemat_m.o elemnt_m.o geom_m.o reg_m.o
```

```
	f90 $(EIGER_FLAGS) -o asmbly_m.o asmbly_m.f90
elevec_m.o: elevec_m.f90 p_potent.o p_file.o excit_m.o basis_m.o elemnt_m.o \
	reg_m.o gauss_m.o vector_m.o cnstnt_m.o iounit_m.o
	f90 $(EIGER_FLAGS) -o elevec_m.o elevec_m.f90
vecasy_m.o: vecasy_m.f90 elevec_m.o excit_m.o geom_m.o elemnt_m.o iounit_m.o
	f90 $(EIGER_FLAGS) -o vecasy_m.o vecasy_m.f90
matrix_m.o: matrix_m.f90
	f90 $(EIGER_FLAGS) -o matrix_m.o matrix_m.f90
load_m.o: load_m.f90 geom_m.o elemnt_m.o cnstnt_m.o
	f90 $(EIGER_FLAGS) -o load_m.o load_m.f90
solutn_m.o: solutn_m.f90 matrix_m.o vecasy_m.o asmbly_m.o excit_m.o load_m.o \
	gauss_m.o cnstnt_m.o iounit_m.o fsseiger_m.o
	f90 $(EIGER_FLAGS) -o solutn_m.o solutn_m.f90
eiger.o: eiger.f90 solutn_m.o excit_m.o geom_m.o elemnt_m.o cnstnt_m.o \
	filemod.o iounit_m.o numparm.o
	f90 $(EIGER_FLAGS) -o eiger.o eiger.f90
curvpt.o: curvpt.f
	f90 $(EIGER_FLAGS) -o curvpt.o curvpt.f
fmin.o: fmin.f
	f90 $(EIGER_FLAGS) -o fmin.o fmin.f
initr.o: initr.f
	f90 $(EIGER_FLAGS) -o initr.o initr.f
lnsrch.o: lnsrch.f
	f90 $(EIGER_FLAGS) -o lnsrch.o lnsrch.f
lubksb.o: lubksb.f
	f90 $(EIGER_FLAGS) -o lubksb.o lubksb.f
ludcmp.o: ludcmp.f
	f90 $(EIGER_FLAGS) -o ludcmp.o ludcmp.f
newt.o: newt.f
	f90 $(EIGER_FLAGS) -o newt.o newt.f
solver.o: solver.f
	f90 $(EIGER_FLAGS) -o solver.o solver.f
```

```
pproj.o: pproj.f
	f90 $(EIGER_FLAGS) -o pproj.o pproj.f
stript.o: stript.f
	f90 $(EIGER_FLAGS) -o stript.o stript.f
triapt.o: triapt.f
	f90 $(EIGER_FLAGS) -o triapt.o triapt.f
tripar.o: tripar.f
	f90 $(EIGER_FLAGS) -o tripar.o tripar.f
cerfc.o: cerfc.f
	f90 $(EIGER_FLAGS) -o cerfc.o cerfc.f

eiger.db: $(EIGER_DB_DEP)
	f90 -o eiger.db -g -check underflow -ladebug $(EIGER_DB_DEP)
fsseiger_m.do: fsseiger_m.f90 matrix_m.do iounit_m.do vector_m.do geom_m.do \
	  excit_m.do reg_m.do elevec_m.do cnstnt_m.do vecasy_m.do
	f90 $(EIGER_DB_FLAGS) -o fsseiger_m.do fsseiger_m.f90
numparm.do: numparm.f90 reg_m.do iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o numparm.do numparm.f90
filemod.do: filemod.f90
	f90 $(EIGER_DB_FLAGS) -o filemod.do filemod.f90
gltramod.do: gltramod.f90
	f90 $(EIGER_DB_FLAGS) -o gltramod.do gltramod.f90
utilmod.do: utilmod.f90
	f90 $(EIGER_DB_FLAGS) -o utilmod.do utilmod.f90
iounit_m.do: iounit_m.f90
	f90 $(EIGER_DB_FLAGS) -o iounit_m.do iounit_m.f90
cnstnt_m.do: cnstnt_m.f90
	f90 $(EIGER_DB_FLAGS) -o cnstnt_m.do cnstnt_m.f90
vector_m.do: vector_m.f90
	f90 $(EIGER_DB_FLAGS) -o vector_m.do vector_m.f90
sylpy_m.do: sylpy_m.f90 vector_m.do
	f90 $(EIGER_DB_FLAGS) -o sylpy_m.do sylpy_m.f90
```

```
gauss_m.do: gauss_m.f90 cnstnt_m.do iounit_m.do numparm.do
	f90 $(EIGER_DB_FLAGS) -o gauss_m.do gauss_m.f90
linear_m.do: linear_m.f90 vector_m.do iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o linear_m.do linear_m.f90
reg_m.do: reg_m.f90 vector_m.do cnstnt_m.do gltramod.do iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o reg_m.do reg_m.f90
elemnt_m.do: elemnt_m.f90 reg_m.do linear_m.do sylpy_m.do vector_m.do \
	 cnstnt_m.do iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o elemnt_m.do elemnt_m.f90
geom_m.do: geom_m.f90 elemnt_m.do iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o geom_m.do geom_m.f90
basis_m.do: basis_m.f90 elemnt_m.do cnstnt_m.do
	f90 $(EIGER_DB_FLAGS) -o basis_m.do basis_m.f90
tubept.do: tubept.f90 vector_m.do
	f90 $(EIGER_DB_FLAGS) -o tubept.do tubept.f90
surfpt_m.do: surfpt_m.f90 elemnt_m.do gauss_m.do vector_m.do cnstnt_m.do \
	 iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o surfpt_m.do surfpt_m.f90
juncpt_m.do: juncpt_m.f90 surfpt_m.do elemnt_m.do gauss_m.do vector_m.do \
	 cnstnt_m.do iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o juncpt_m.do juncpt_m.f90
intrfc_m.do: intrfc_m.f90 surfpt_m.do tubept.do elemnt_m.do vector_m.do \
	 iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o intrfc_m.do intrfc_m.f90
excit_m.do: excit_m.f90 reg_m.do vector_m.do cnstnt_m.do iounit_m.do utilmod.do
	f90 $(EIGER_DB_FLAGS) -o excit_m.do excit_m.f90
p_file.do: p_file.f90 reg_m.do cnstnt_m.do
	f90 $(EIGER_DB_FLAGS) -o p_file.do p_file.f90
p_spfun.do: p_spfun.f90
	f90 $(EIGER_DB_FLAGS) -o p_spfun.do p_spfun.f90
p_tline.do: p_tline.f90 p_file.do p_spfun.do
	f90 $(EIGER_DB_FLAGS) -o p_tline.do p_tline.f90
```

```
p_period.do: p_period.f90 elemnt_m.do vector_m.do numparm.do
	f90 $(EIGER_DB_FLAGS) -o p_period.do p_period.f90
p_somint.do: p_somint.f90 p_tline.do p_spfun.do p_file.do vector_m.do \
	 iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o p_somint.do p_somint.f90
p_potent.do: p_potent.f90 p_somint.do p_period.do p_tline.do p_spfun.do \
	 p_file.do excit_m.do juncpt_m.do intrfc_m.do elemnt_m.do \
	 linear_m.do gauss_m.do vector_m.do
	f90 $(EIGER_DB_FLAGS) -o p_potent.do p_potent.f90
cmptpt_m.do: cmptpt_m.f90 p_potent.do p_file.do juncpt_m.do intrfc_m.do \
	 elemnt_m.do gauss_m.do vector_m.do cnstnt_m.do \
	 iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o cmptpt_m.do cmptpt_m.f90
elemat_m.do: elemat_m.f90 cmptpt_m.do basis_m.do elemnt_m.do gauss_m.do \
	 vector_m.do cnstnt_m.do iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o elemat_m.do elemat_m.f90
asmbly_m.do: asmbly_m.f90 elemat_m.do elemnt_m.do geom_m.do reg_m.do
	f90 $(EIGER_DB_FLAGS) -o asmbly_m.do asmbly_m.f90
elevec_m.do: elevec_m.f90 p_potent.do p_file.do excit_m.do basis_m.do \
	 elemnt_m.do reg_m.do gauss_m.do vector_m.do cnstnt_m.do \
	 iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o elevec_m.do elevec_m.f90
vecasy_m.do: vecasy_m.f90 elevec_m.do excit_m.do geom_m.do elemnt_m.do \
	 iounit_m.do
	f90 $(EIGER_DB_FLAGS) -o vecasy_m.do vecasy_m.f90
matrix_m.do: matrix_m.f90
	f90 $(EIGER_DB_FLAGS) -o matrix_m.do matrix_m.f90
load_m.do: load_m.f90 geom_m.do elemnt_m.do cnstnt_m.do
	f90 $(EIGER_DB_FLAGS) -o load_m.do load_m.f90
solutn_m.do: solutn_m.f90 matrix_m.do vecasy_m.do asmbly_m.do excit_m.do \
	 load_m.do gauss_m.do cnstnt_m.do iounit_m.do fsseiger_m.do
	f90 $(EIGER_DB_FLAGS) -o solutn_m.do solutn_m.f90
```

```
eiger.do: eiger.f90 solutn_m.do excit_m.do geom_m.do elemnt_m.do cnstnt_m.do \
        filemod.do iounit_m.do numparm.do
          f90 $(EIGER_DB_FLAGS) -o eiger.do eiger.f90
curvpt.do: curvpt.f
          f90 $(EIGER_DB_FLAGS) -o curvpt.do curvpt.f
fmin.do: fmin.f
          f90 $(EIGER_DB_FLAGS) -o fmin.do fmin.f
initr.do: initr.f
          f90 $(EIGER_DB_FLAGS) -o initr.do initr.f
lnsrch.do: lnsrch.f
          f90 $(EIGER_DB_FLAGS) -o lnsrch.do lnsrch.f
lubksb.do: lubksb.f
          f90 $(EIGER_DB_FLAGS) -o lubksb.do lubksb.f
ludcmp.do: ludcmp.f
          f90 $(EIGER_DB_FLAGS) -o ludcmp.do ludcmp.f
newt.do: newt.f
          f90 $(EIGER_DB_FLAGS) -o newt.do newt.f
solver.do: solver.f
          f90 $(EIGER_DB_FLAGS) -o solver.do solver.f
pproj.do: pproj.f
          f90 $(EIGER_DB_FLAGS) -o pproj.do pproj.f
stript.do: stript.f
          f90 $(EIGER_DB_FLAGS) -o stript.do stript.f
triapt.do: triapt.f
          f90 $(EIGER_DB_FLAGS) -o triapt.do triapt.f
tripar.do: tripar.f
          f90 $(EIGER_DB_FLAGS) -o tripar.do tripar.f
cerfc.do: cerfc.f
          f90 $(EIGER_DB_FLAGS) -o cerfc.do cerfc.f

clean:
          rm *.mod *.o *.do
```