

RF PCB Toolbox™

Reference



MATLAB®

R2022b



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

RF PCB Toolbox™ Reference

© COPYRIGHT 2021–2022 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

September 2021	Online only	New for Version 1.0 (R2021b)
March 2022	Online only	Revised for Version 1.1 (R2022a)
September 2022	Online only	Revised for Version 1.2 (R2022b)

1 | Objects

2 | Functions

Objects

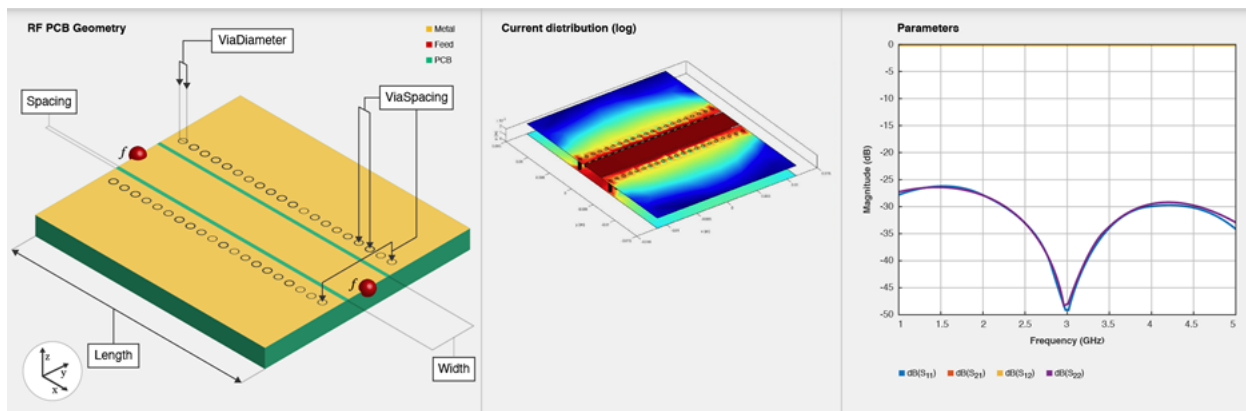
coplanarWaveguide

Create coplanar waveguide transmission line

Description

Use the `coplanarWaveguide` object to create a coplanar waveguide transmission line. A Coplanar waveguide is a common type of transmission line used in any PCB implementation of RF and microwave components. A coplanar waveguide transmission line has a center conductor strip and two ground planes. One ground plane is the layer that acts as the conductor strip and the other ground plane is the bottom layer.

Note This PCB object supports behavioral modeling. For more information, see “Behavioral Models”. To analyze the behavioral model for a coplanar waveguide, set the `Behavioral` property in the `sparameters` function to `true` or `1`



Creation

Syntax

```
cpgw = coplanarWaveguide
cpgw = coplanarWaveguide(Name=Value)
```

Description

`cpgw = coplanarWaveguide` creates a default coplanar waveguide transmission line with a Teflon substrate. The default property values are for a 50 ohm transmission line.

`cpgw = coplanarWaveguide(Name=Value)` sets “Properties” on page 1-3 using one or more name-value arguments. For example, `coplanarWaveguide(Width=0.0047)` creates a coplanar waveguide transmission line of width 0.0047 meters. Properties not specified retain their default values.

Properties

Length — Length of coplanar waveguide transmission line

0.0231 (default) | positive scalar

Length of the coplanar waveguide transmission line in meters, specified as a positive scalar.

Example: `cpgw = coplanarWaveguide(Length=0.0300)`

Data Types: double

Width — Width of coplanar waveguide transmission line

0.0039 (default) | positive scalar

Width of the coplanar waveguide transmission line in meters, specified as a positive scalar.

Example: `cpgw = coplanarWaveguide(Width=0.0047)`

Data Types: double

Spacing — Distance between transmission line and adjacent ground plane

2.0000e-04 (default) | positive scalar

Distance between the transmission line and the adjacent top layer metal of the ground plane, specified as a positive scalar in meters.

Example: `cpgw = coplanarWaveguide(Spacing=3.0000e-04)`

Data Types: double

ViaSpacing — Distance between vias

[0.0011 0.0070] (default) | two-element vector

Distance between the vias in meters, specified as a two-element vector of positive elements.

Example: `cpgw = coplanarWaveguide(ViaSpacing=[0.0021 0.0060])`

Data Types: double

ViaDiameter — Diameter of via

5.0000e-04 (default) | positive scalar

Diameter of the via in meters, specified as a positive scalar.

Example: `cpgw = coplanarWaveguide(ViaDiameter=7.0000e-04)`

Data Types: double

Height — Height of coplanar waveguide transmission line

0.0016 (default) | positive scalar

Height of the coplanar waveguide transmission line from the ground plane, specified as a positive scalar in meters.

Example: `cpgw = coplanarWaveguide(Height=0.0020)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0300 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `cpgw = coplanarWaveguide(GroundPlaneWidth=0.0350)`

Data Types: double

Substrate — Type of dielectric material

'Teflon' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object.

Example: `d = dielectric("FR4"); cpgw = coplanarWaveguide(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'PEC' (default) | metal object

Type of metal used in the conducting layers, specified as a metal object.

Example: `m = metal("PEC"); cpgw = coplanarWaveguide(Conductor=m)`

Data Types: string | char

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>design</code>	Design coplanar waveguide transmission line around particular frequency
<code>feedCurrent</code>	Calculate current at feed port
<code>getZ0</code>	Calculate characteristic impedance of transmission line
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>shapes</code>	Extract all metal layer shapes of PCB component
<code>show</code>	Display PCB component structure or PCB shape
<code>sparameters</code>	Calculate S-parameters for RF PCB objects

Examples

Create Default Coplanar Waveguide

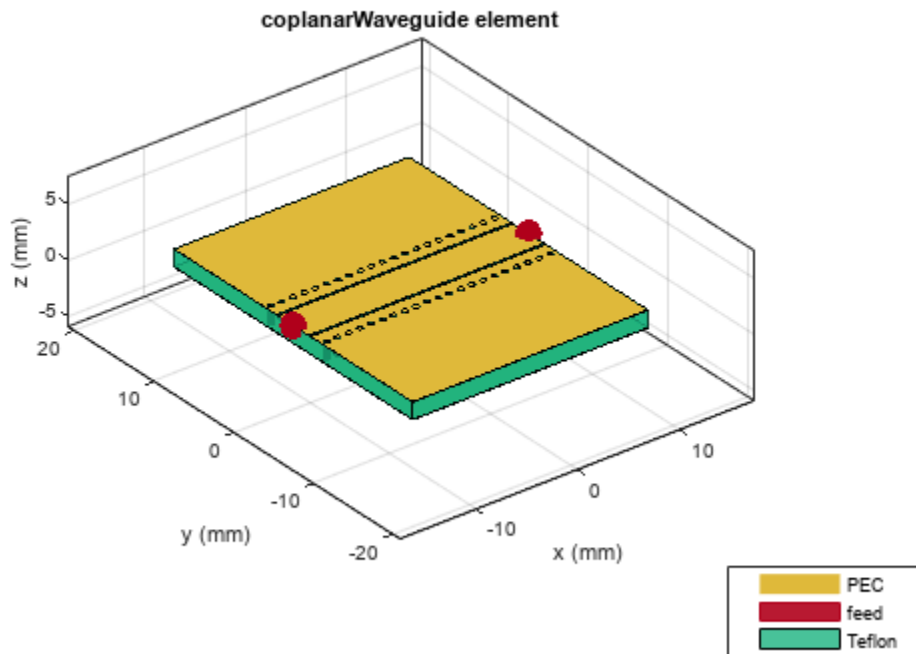
Create a coplanar waveguide transmission line.

```
waveguide = coplanarWaveguide
```

```
waveguide =  
    coplanarWaveguide with properties:  
  
        Length: 0.0231  
        Width: 0.0039  
        Spacing: 2.0000e-04  
        ViaSpacing: [0.0011 0.0070]  
        ViaDiameter: 5.0000e-04  
        Height: 0.0016  
        GroundPlaneWidth: 0.0300  
        Substrate: [1x1 dielectric]  
        Conductor: [1x1 metal]
```


View the coplanar waveguide transmission line.

```
show(waveguide)
```



Calculate the S-parameters of the waveguide from 1-10 GHz.

```
sparam = sparameters(waveguide,1e9:0.3e9:10e9)
```

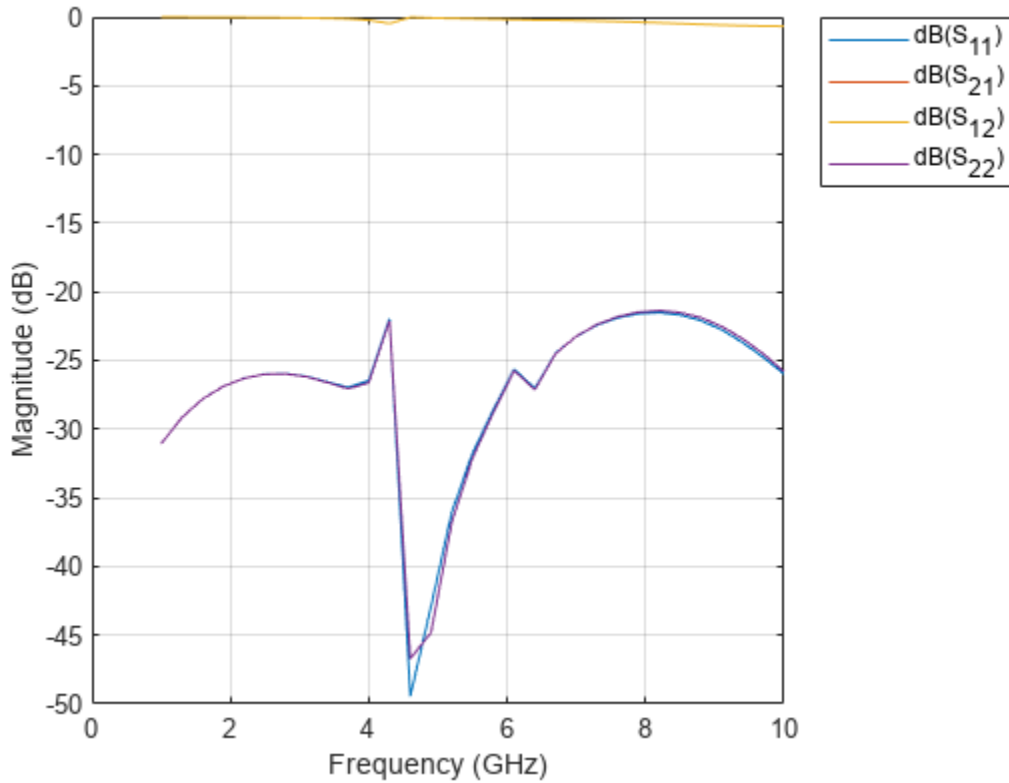
```
sparam =  
  sparameters: S-parameters object
```

```
  NumPorts: 2  
  Frequencies: [31x1 double]  
  Parameters: [2x2x31 double]  
  Impedance: 50
```

```
rfparam(obj,i,j) returns S-parameter  $S_{ij}$ 
```

Plot the S-parameters.

```
rfplot(sparam)
```



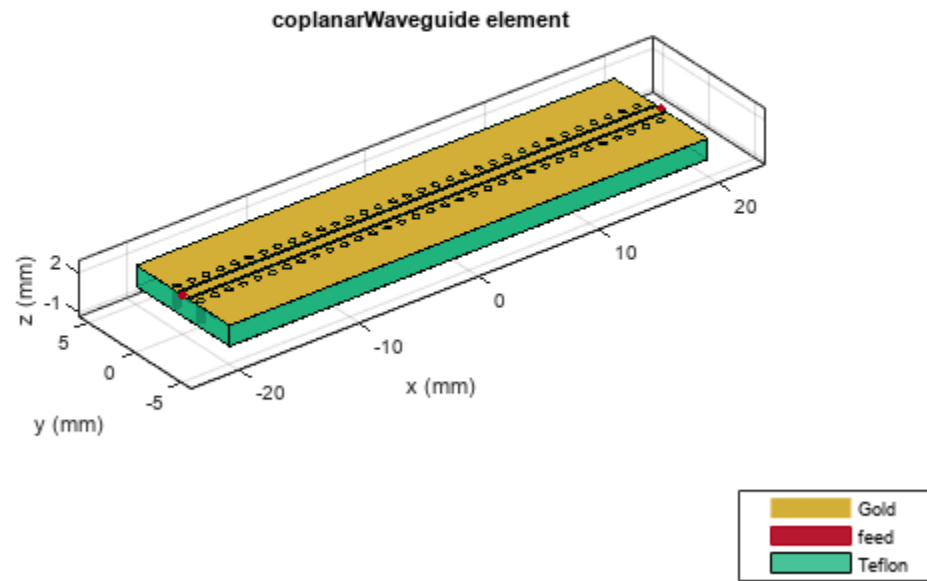
Behavioral S-parameters of Coplanar Waveguide Transmission Line

Create a coplanar waveguide transmission line using a gold substrate as the dielectric.

```
txem = coplanarWaveguide;
txem.Conductor = metal("Gold");
```

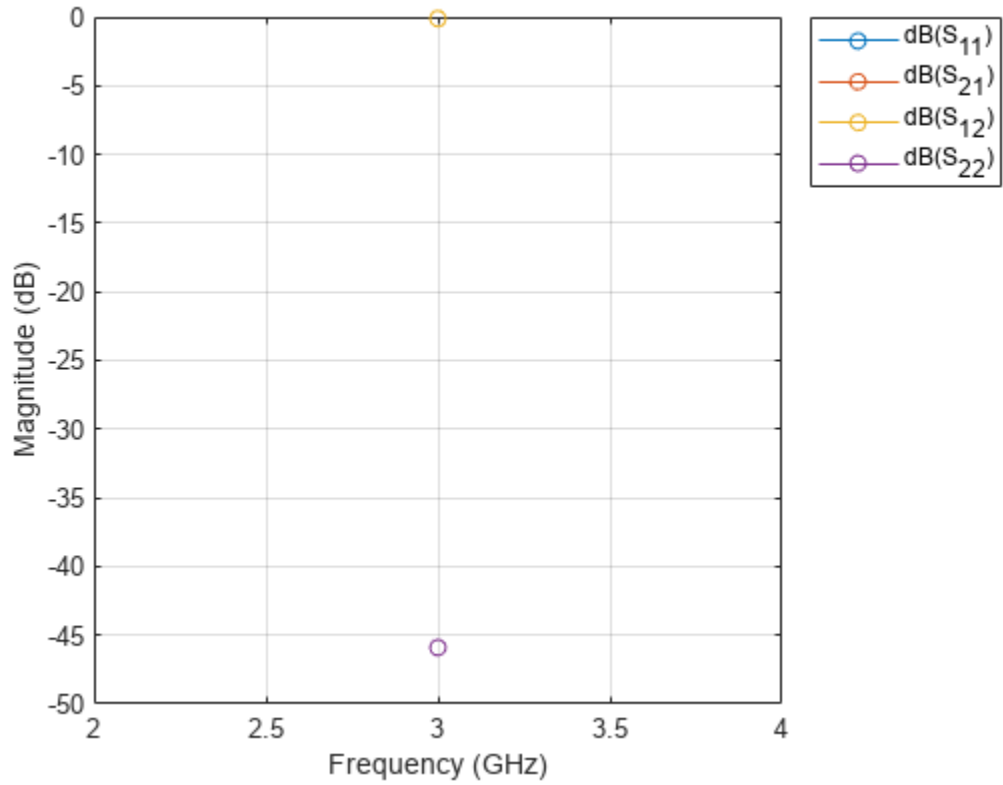
Design the coplanar waveguide at a frequency of 3 GHz, line length of 0.5 meters, and impedance of 75 ohms.

```
txem = design(txem,3e9,LineLength=0.5,Z0=75);
show(txem)
```



Compute and plot the behavioral S-parameters of the waveguide.

```
spar = sparameters(txem,3e9,Behavioral=true);  
rfplot(spar)
```



Version History

Introduced in R2021b

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

microstripLine | coupledMicrostripLine

microstripLine

Create transmission line in microstrip form

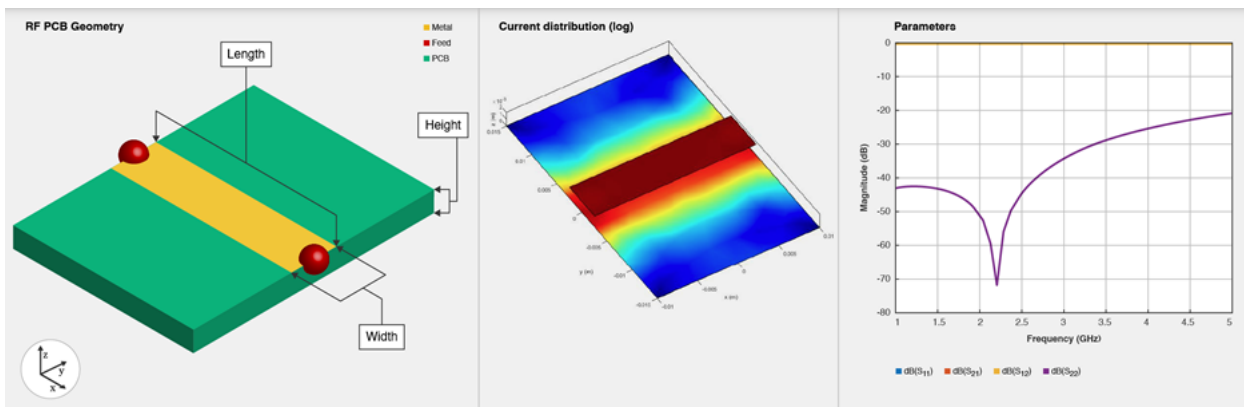
Description

Use the `microstripLine` object to create a microstrip transmission line. A microstrip line is a transmission line that is a basic building block for most RF planar microwave devices. You can use this transmission line to connect two PCB components or to create components such as filters, couplers, and feeding elements of various types of antennas.

Note This PCB object supports behavioral modeling. For more information, see “Behavioral Models”. To analyze the behavioral model for a microstrip transmission line, set the `Behavioral` property in the `parameters` function to `true` or `1`

A few applications of microstrip transmission lines are:

- Creating matching feed and coupling networks
- Transmitting power from one component to another
- Feeding planar antennas and coupling structures
- Creating varying inductances or capacitances using open- or short ended- transmission lines



Creation

Syntax

```
microstrip = microstripLine
microstrip = microstripLine(Name=Value)
microstrip = microstripLine(txlineobj)
```

Description

`microstrip = microstripLine` creates a default microstrip transmission line using a Teflon substrate.

`microstrip = microstripLine(Name=Value)` sets properties using one or more name value pair arguments. For example, `microstrip = microstripLine(Length=0.0300)` creates a microstrip line of length 0.0300 meters. Properties not specified retain their default values.

`microstrip = microstripLine(txlineobj)` creates a microstrip transmission line from the behavioral model of a `txlineMicrostrip` object in RF Toolbox™.

Properties

Length — Length of microstrip line

0.0200 (default) | positive scalar

Length of the microstrip line in meters, specified as a positive scalar.

Example: `microstrip = microstripLine(Length=0.0300)`

Data Types: double

Width — Width of microstrip line

0.0050 (default) | positive scalar

Width of the microstrip line in meters, specified as a positive scalar.

Example: `microstrip = microstripLine(Width=0.00630)`

Data Types: double

Height — Height of microstrip line

0.0016 (default) | positive scalar

Height of the microstrip line from the ground plane in meters, specified as a positive scalar.

In the case of a multilayer substrate, you can use the height property to create a microstrip line at the interface of the two dielectrics.

Example: `microstrip = microstripLine(Height=0.0015)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0300 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `microstrip = microstripLine(GroundPlaneWidth=0.0400)`

Data Types: double

Substrate — Type of dielectric material

'Teflon' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a `dielectric` object. For more information, see `dielectric`. The thickness of the default dielectric material Teflon is 0.0016 m or the same as the height property.

```
Example: d = dielectric('FR4'); microstrip = microstripLine(Substrate=d)
```

Data Types: string | char

Conductor — Type of metal used in conducting layers

'PEC' (default) | metal object

Type of metal used in conducting layers, specified as a `metal` object. For more information see `metal`.

```
Example: m = metal('PEC'); microstrip = microstripLine(Conductor=m)
```

Data Types: string | char

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>design</code>	Design microstrip transmission line around specified frequency
<code>feedCurrent</code>	Calculate current at feed port
<code>getZ0</code>	Calculate characteristic impedance of transmission line
<code>layout</code>	Plot all metal layers and board shape
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>shapes</code>	Extract all metal layer shapes of PCB component
<code>show</code>	Display PCB component structure or PCB shape
<code>sparameters</code>	Calculate S-parameters for RF PCB objects

Examples

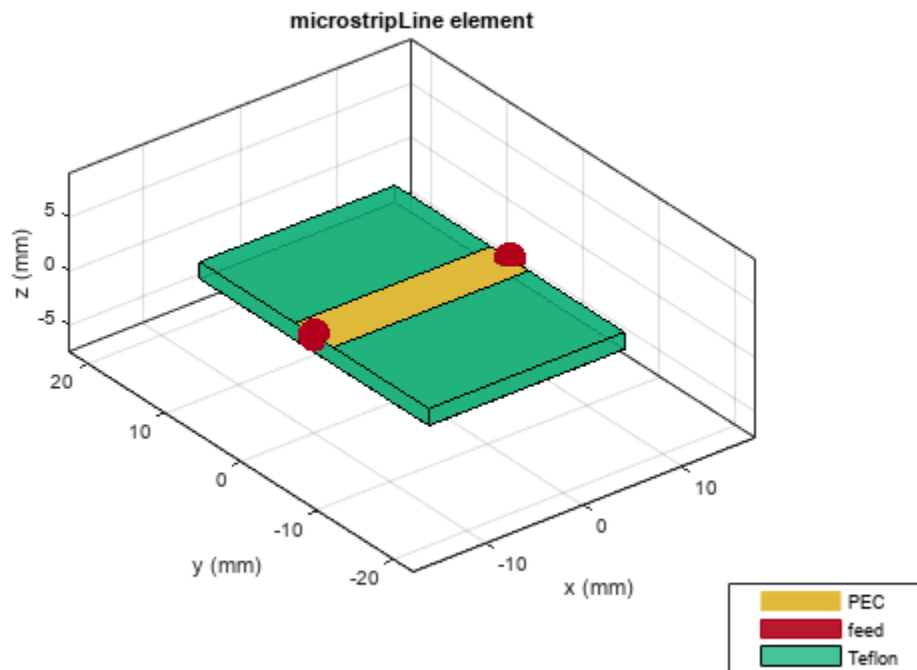
Default Microstrip Line

Create and view a default microstrip transmission line.

```
microstrip = microstripLine

microstrip =
    microstripLine with properties:
        Length: 0.0200
        Width: 0.0050
        Height: 0.0016
        GroundPlaneWidth: 0.0300
        Substrate: [1x1 dielectric]
        Conductor: [1x1 metal]

show(microstrip)
```



Microstrip Transmission Line at 3 GHz

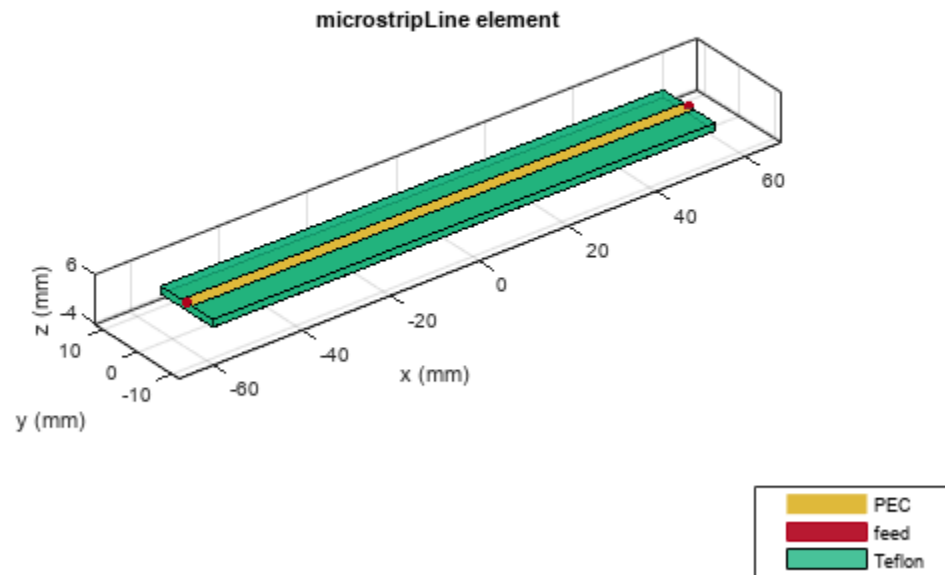
Design a microstrip transmission line at 3 GHz, with a characteristic impedance of 70 ohms and a line length 1.5 times the wavelength.

```
microstrip = design(microstripLine,3e9,'Z0',70,'LineLength',1.5)
```

```
microstrip =  
  microstripLine with properties:  
    Length: 0.1132  
    Width: 0.0030  
    Height: 0.0016  
  GroundPlaneWidth: 0.0150  
    Substrate: [1x1 dielectric]  
    Conductor: [1x1 metal]
```

View the microstrip transmission line.

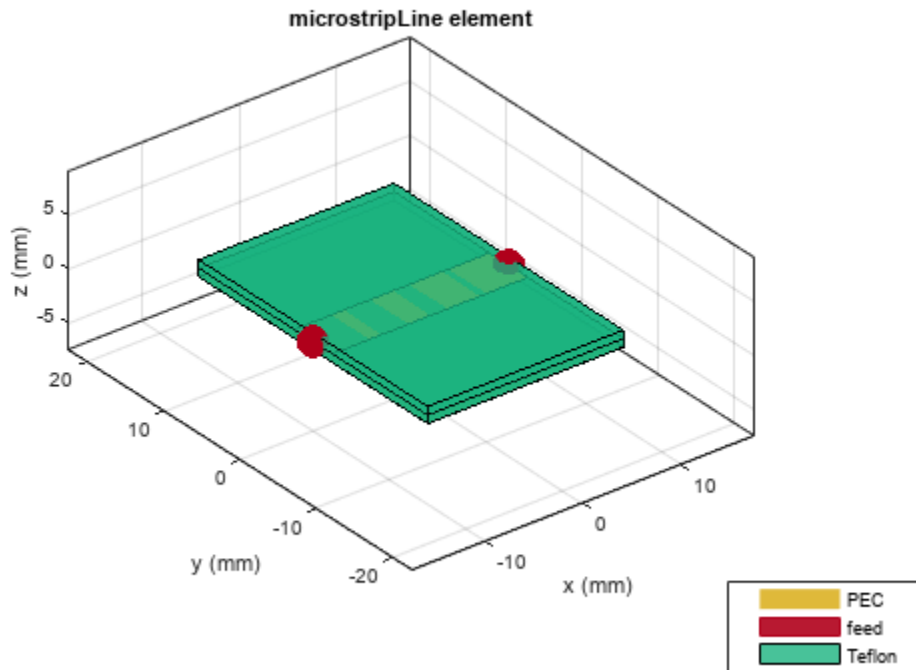
```
show(microstrip)
```

Multilayer Dielectric Microstrip Transmission Line

Create and view a multilayer dielectric microstrip transmission line.

```
microstrip = microstripLine;  
microstrip.Substrate = dielectric('Name',{'Teflon','Teflon'},'EpsilonR', ...  
    [2.1 2.1],'LossTangent',[0 0],'Thickness',[0.8e-3 0.8e-3]);  
microstrip.Height = 0.8e-3;  
show(microstrip);
```



Use Behavioral Model to Calculate S-Parameters of Microstrip Cross

Design a microstrip transmission line at 3 GHz for FR4 substrate.

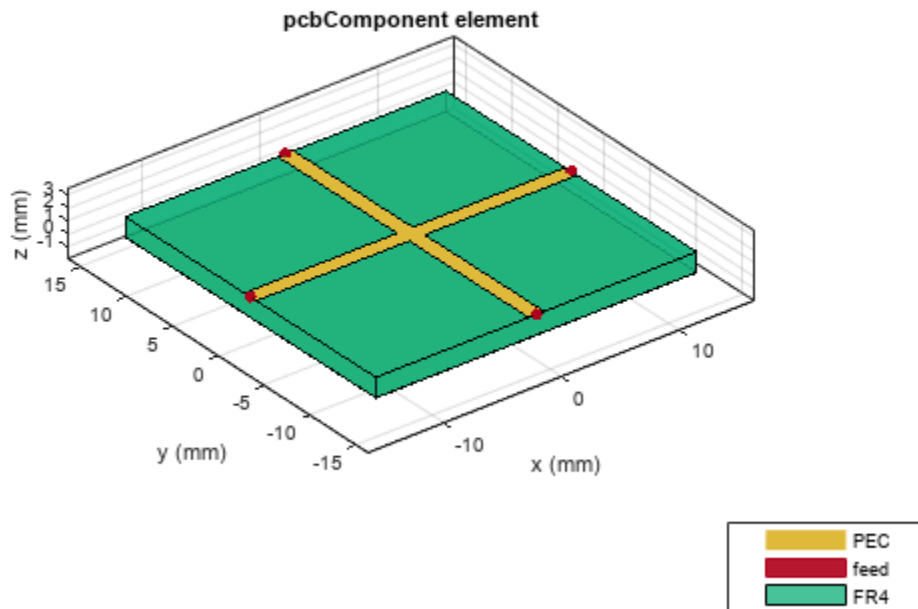
```
d = dielectric('FR4');
d.LossTangent = 0;
m = design(microstripLine('Substrate',d),3e9,'Z0',75,...
    'LineLength',0.5);
```

Create a microstrip cross.

```
layer2d = traceCross('Length',[m.Length m.Length], ...
    'Width',[m.Width m.Width]);
```

Convert the cross trace to a PCB component.

```
robj = pcbComponent(layer2d);
robj.BoardThickness = m.Substrate.Thickness;
robj.Layers{2} = m.Substrate;
show(robj)
```



Define frequency points to calculate the s-parameters.

```
freq = (1:3:40)*100e6;
```

Calculate the s-parameters of the cross trace using the behavioral model.

```
Sckt = sparameters(robj, freq, 'Behavioral', true);
```

Warning: Behavioral model is valid only when EpsilonR is 9.9.

Calculate the s-parameters of the cross trace using the electromagnetic solver.

```
Sem = sparameters(robj, freq);
```

References:

- 1 Ramesh Garg & I. J. Bahl (1978) Microstrip discontinuities, International Journal of Electronics, 45:1, 81-87, DOI: [10.1080/00207217808900883](https://doi.org/10.1080/00207217808900883)
- 2 Wadell, Brian C. *Transmission Line Design Handbook*. The Artech House Microwave Library. Boston: Artech House, 1991.

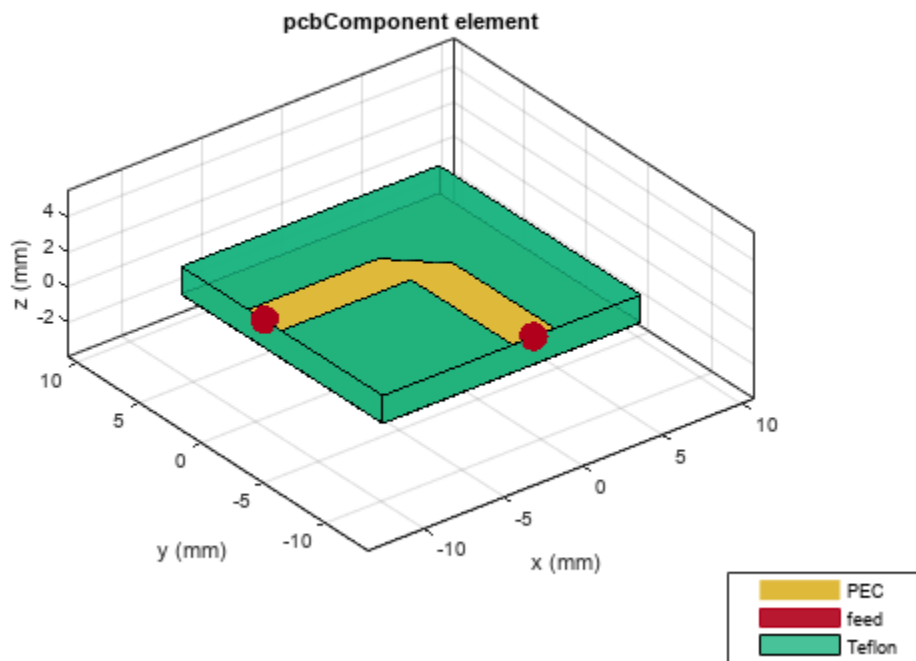
Use Behavioral Model to Calculate S-Parameters of Mitered Bend Microstrip

Create mitered bend microstrip.

```

m = design(microstripLine,6e9,"Z0",75);
layer2d = bendMitered('Length',[m.Length/2 m.Length/2],...
"Width",[m.Width m.Width],'MiterDiagonal',sqrt(2)*m.Width);
robj = pcbComponent(layer2d);
robj.BoardThickness = m.Substrate.Thickness;
robj.Layers{2} = m.Substrate;
show(robj)

```

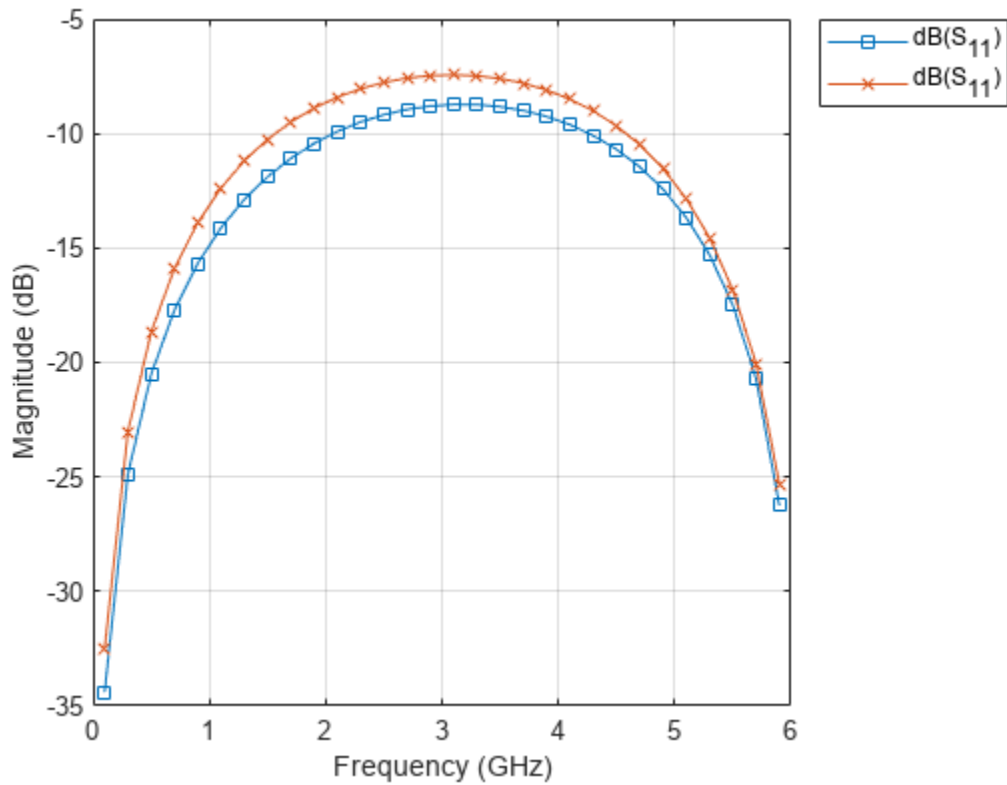


Compute and plot s-parameters.

```

freq = (1:2:60)*100e6;
Sckt = sparameters(robj,freq,'Behavioral',true);
Sem = sparameters(robj,freq);
rfplot(Sckt,1,1,'db','-s')
hold on
rfplot(Sem,1,1,'db','-x')

```



Reference:

M. Kirschning, R. H. Jansen and N. H. L. Koster, "Measurement and Computer-Aided Modeling of Microstrip Discontinuities by an Improved Resonator Method," 1983 IEEE MTT-S International Microwave Symposium Digest, Boston, MA, USA, 1983, pp. 495-497, doi: 10.1109/MWSYM.1983.1130959.

Version History

Introduced in R2021b

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

`coplanarWaveguide` | `coupledMicrostripLine`

spiralInductor

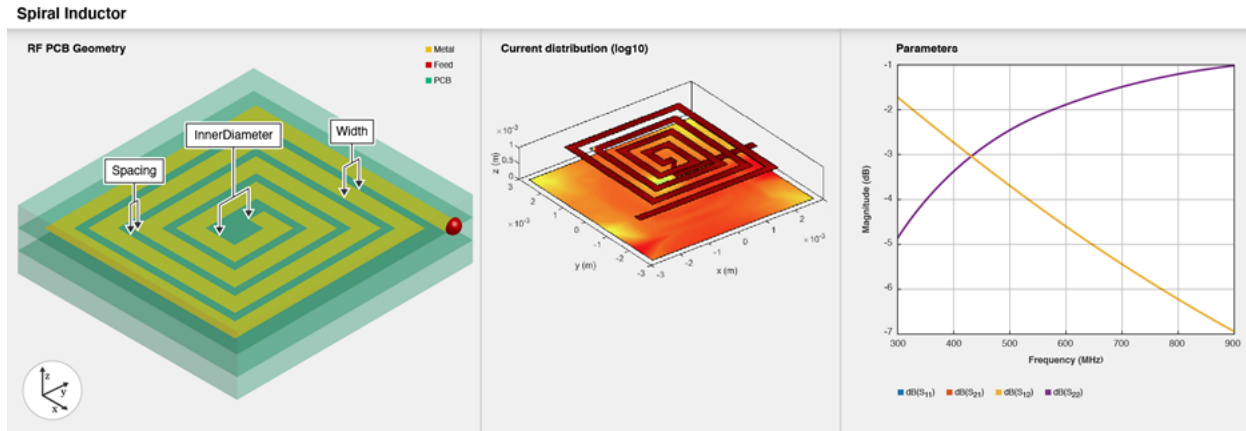
Create spiral inductor in four different shapes

Description

Use the `spiralInductor` object to create a spiral inductor in one of four different shapes: square, circle, hexagon, or octagon. The spiral inductor is a two-port planar inductor with a single or multiple dielectric layers. A turn in a spiral inductor is the length of a complete 360-degree revolution. Spiral inductor filaments have uniform spacing and width throughout the structure. Spiral inductors are an integral part of many radio-frequency and microwave circuits, acting as resonant elements or chokes. The inductor feed can be configured in one of the following two ways:

Note This PCB object supports behavioral modeling. For more information, see “Behavioral Models”. To analyze the behavioral model for a spiral inductor, set the `Behavioral` property in the `sparameters` function to `true` or `1`

- The input and output ports are punched through at the same layer.
- The input port is routed out from the layer below the inductor by a via hole. The output port is extended to the end of the dielectric in the same layer.



Creation

Syntax

```
inductor = spiralInductor
inductor = spiralInductor(Name=Value)
```

Description

`inductor = spiralInductor` creates a square spiral planar inductor. The default properties are for a resonant frequency of 600 MHz.

`inductor = spiralInductor(Name=Value)` sets "Properties" on page 1-19 using one or more name-value arguments. For example, `spiralInductor(SpiralShape="Octagon")` creates an octagonal spiral inductor. Properties not specified retain their default values.

Properties

SpiralShape — Shape of spiral inductor

"Square" (default) | "Circle" | "Hexagon" | "Octagon"

Shape of the spiral inductor, specified as either "Square", "Circle", "Hexagon", or "Octagon".

Example: `inductor = spiralInductor(SpiralShape="Circle")`

Data Types: string | char

InnerDiameter — Inner diameter of polygon along edge

5.0000e-04 (default) | positive scalar

Inner diameter of the polygon along the edge in meters, specified as a positive scalar.

Example: `inductor = spiralInductor(InnerDiameter=8.0000e-04)`

Data Types: double

Width — Strip width

2.5000e-04 (default) | positive scalar

Strip width in meters, specified as a positive scalar.

Example: `inductor = spiralInductor(Width=3.8000e-04)`

Data Types: double

Spacing — Distance between strips

2.5000e-04 (default) | positive scalar

Distance between the strips in meters, specified as a positive scalar.

Example: `inductor = spiralInductor(Spacing=3.8000e-04)`

Data Types: double

NumTurns — Number of turns in spiral inductor

4 (default) | positive scalar

Number of turns in the spiral inductor, specified as a positive scalar. You can specify a minimum of 1 turn and a maximum of 12 turns. One turn length is the length of a complete 360-degree revolution.

Example: `inductor = spiralInductor(NumTurns=6)`

Data Types: double

Height — Height from ground plane to inductor

0.0010 (default) | positive scalar

Height from the ground plane to the inductor in meters, specified as a positive scalar.

Example: `inductor = spiralInductor(Height=0.0056)`

Data Types: double

GroundPlaneLength — Length of ground plane`0.0056 (default) | positive scalar`

Length of the ground plane in meters, specified as a positive scalar. This object does not support infinite ground plane length.

Example: `inductor = spiralInductor(GroundPlaneLength=0.046)`

Example: `double`

GroundPlaneWidth — Width of ground plane`0.0056 (default) | positive scalar`

Width of the ground plane in meters, specified as a positive scalar. This object does not support infinite ground plane width.

Example: `inductor = spiralInductor(GroundPlaneWidth=0.046)`

Example: `double`

Substrate — Type of dielectric material`dielectric object`

Type of dielectric material used as a substrate, specified as a dielectric object. The default value is a dielectric object with these properties:

- `Name`—{'RTDuroid', 'RTDuroid', 'RTDuroid'}
- `EpsilonR`—[3.66, 3.66, 3.66]
- `LossTangent`—[0.0013, 0.0013, 0.0013]
- `Thickness`—[0.508e-3, 0.508e-3, 0.508e-3]

Example: `d = dielectric("FR4"); inductor = spiralInductor(Substrate=d)`

Data Types: `string | char`

Conductor — Type of metal used in conducting layers`'Copper' (default) | metal object`

Type of metal used in the conducting layers, specified as a metal object.

Example: `m = metal("PEC"); inductor = spiralInductor(Conductor=m)`

Data Types: `string | char`

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>feedCurrent</code>	Calculate current at feed port
<code>getZ0</code>	Calculate characteristic impedance of transmission line
<code>inductance</code>	Calculate inductance
<code>layout</code>	Plot all metal layers and board shape
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>shapes</code>	Extract all metal layer shapes of PCB component
<code>show</code>	Display PCB component structure or PCB shape
<code>sparameters</code>	Calculate S-parameters for RF PCB objects

Examples

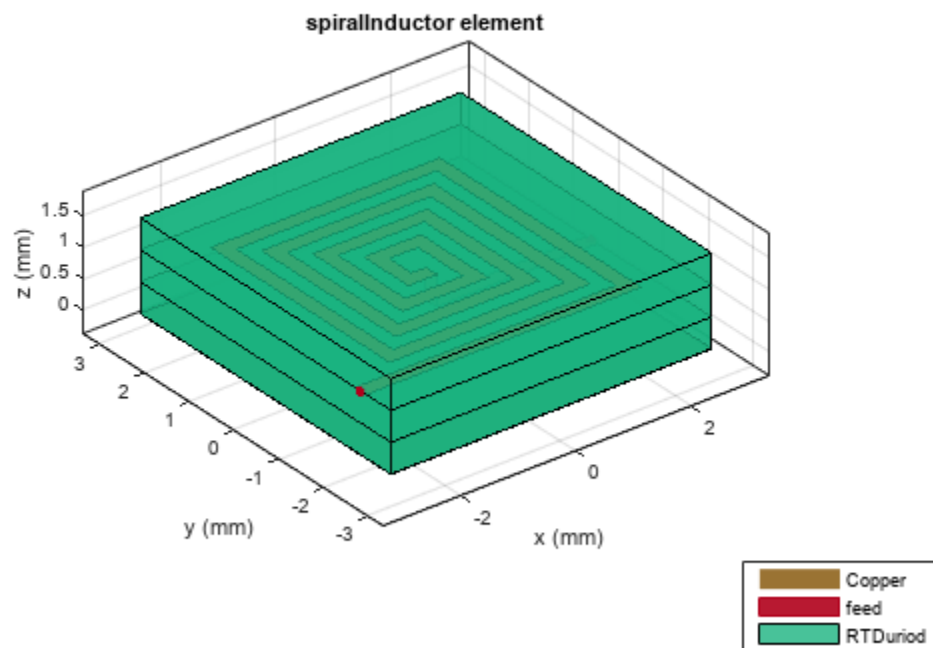
Create Default Spiral Inductor

Create and view a default spiral inductor.

```
inductor = spiralInductor
```

```
inductor =  
    spiralInductor with properties:  
  
        SpiralShape: 'Square'  
        InnerDiameter: 5.0000e-04  
        Width: 2.5000e-04  
        Spacing: 2.5000e-04  
        NumTurns: 4  
        Height: 0.0010  
        GroundPlaneLength: 0.0056  
        GroundPlaneWidth: 0.0056  
        Substrate: [1x1 dielectric]  
        Conductor: [1x1 metal]
```

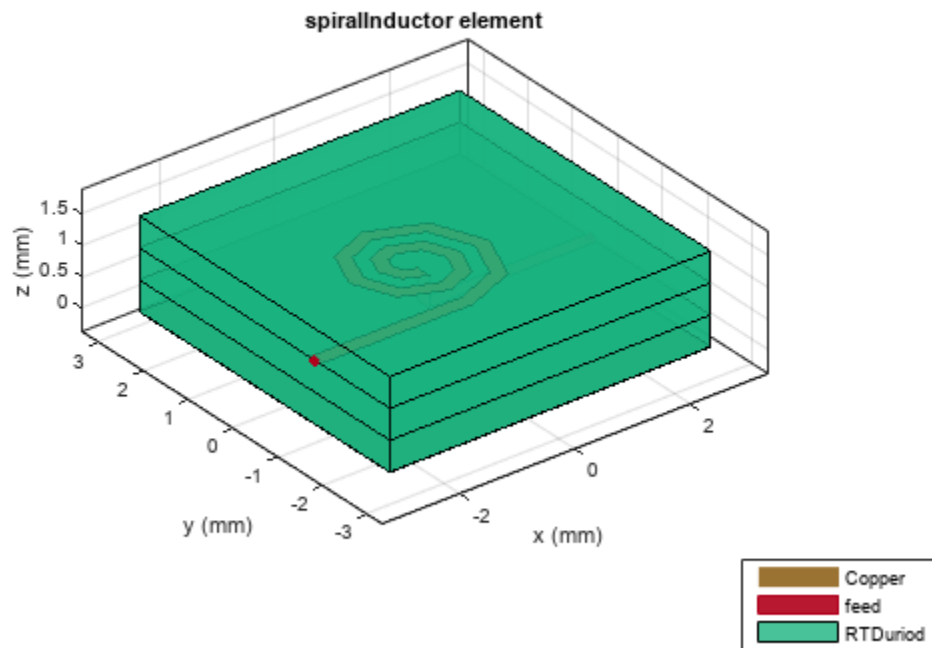
```
show(inductor)
```



Octagonal Spiral Inductor

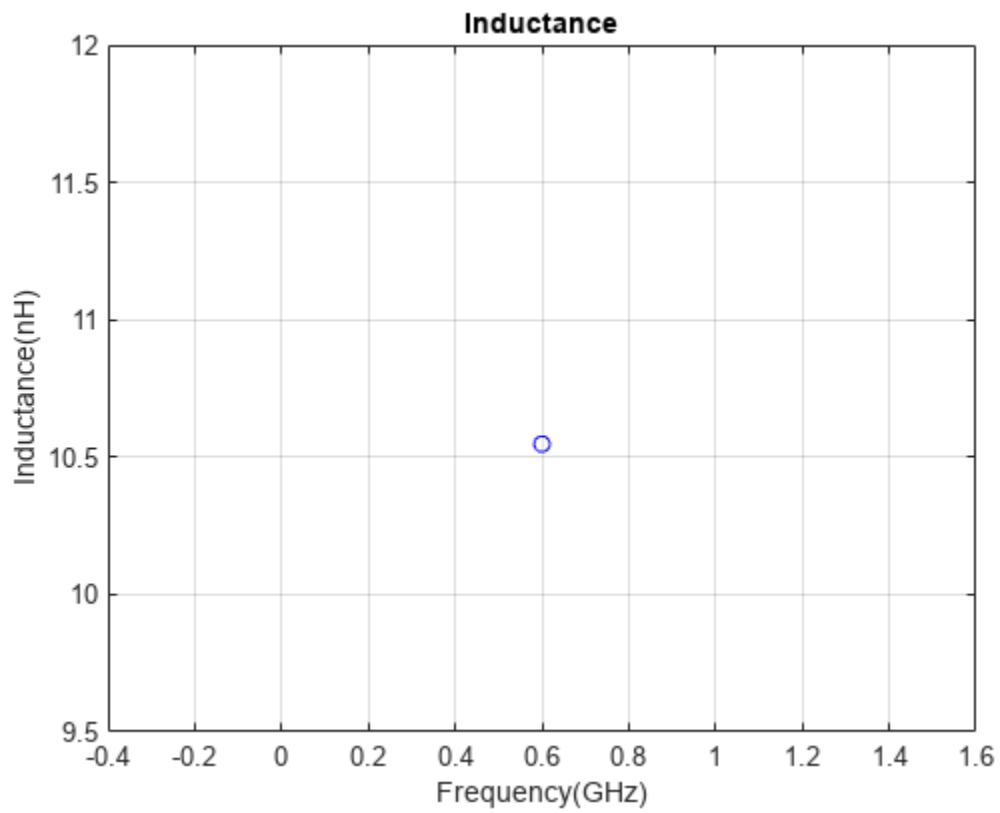
Create and view a two-turn octagonal spiral inductor.

```
inductor = spiralInductor(SpiralShape="Octagon",NumTurns=2);  
show(inductor)
```



Measure the inductance of the inductor.

```
figure  
inductance(inductor,600e6)
```



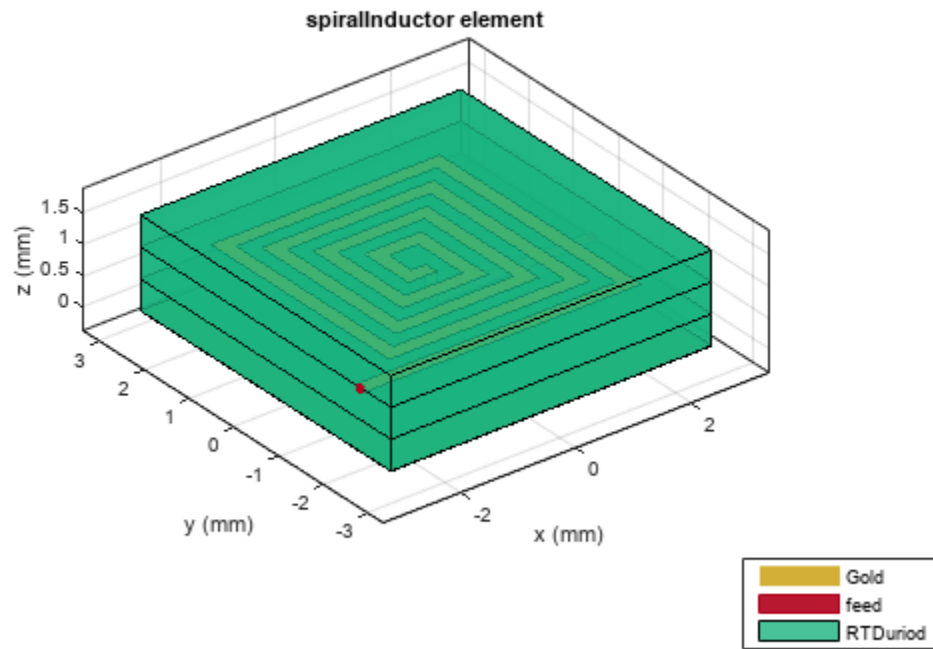
```
ind = inductance(inductor,600e6)
```

```
ind = 1.0547e-08
```

Analyze Spiral Inductor Using Behavioral S-parameters

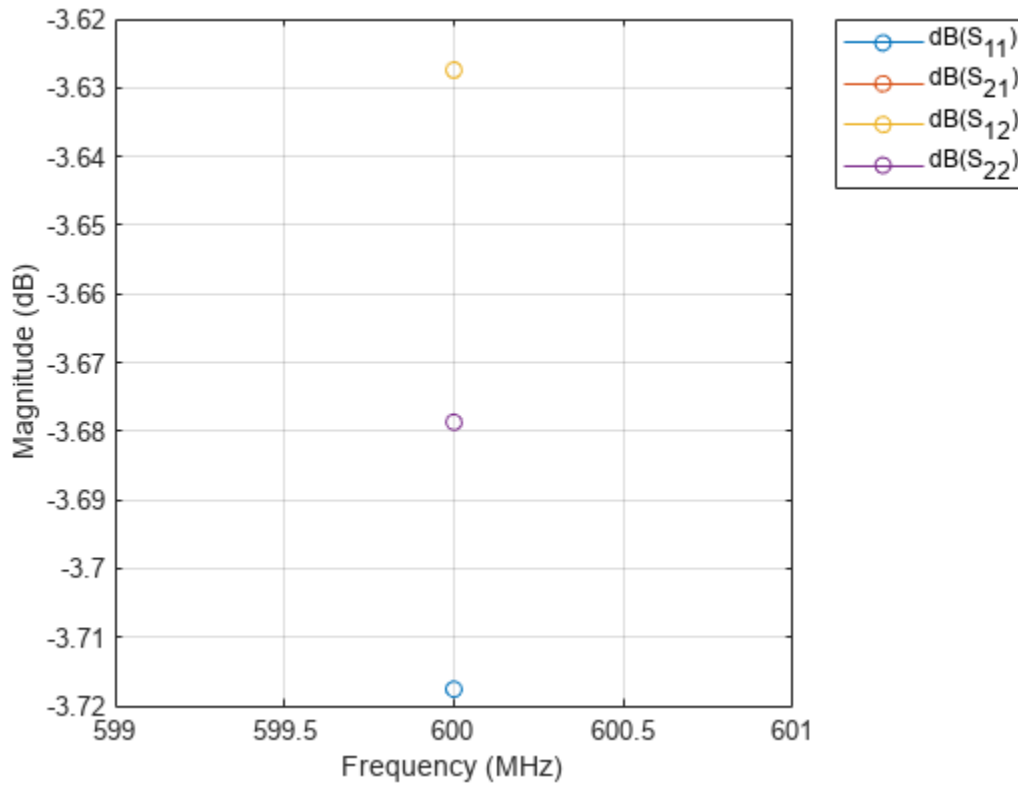
Create a spiral inductor using gold as the conductor.

```
inductor = spiralInductor;  
inductor.Conductor = metal("Gold");  
show(inductor)
```



Compute and plot the behavioral S-parameters of the inductor at 600 MHz.

```
spar = sparameters(inductor,600e6,Behavioral=true);  
rfplot(spar)
```



More About

Parametric Analysis Guidelines

- The inductance of a spiral inductor is directly proportional to the physical parameters such as NumTurns, Spacing, and Width.
- The decrease in the width reduces the capacitance between the turns of the inductor.

Inductor Length and Area

Inductor length equation:

$$inductorlength = NumTurns * D_{avg} * N * \tan\left(\frac{\pi}{N}\right)$$

where,

- N = Number of sides of the polygon
- $D_{avg} = \frac{(d_{out} - d_{in})}{(d_{out} + d_{in})}$

where when `obj = spiralInductor`,

$$d_{in} = obj.InnerDiameter$$

$$d_{out} = obj.InnerDiameter + 2((obj.NumTurns * obj.Width) + (obj.NumTurns - 1) * obj.Spacing)$$

Version History

Introduced in R2021b

References

- [1] Beerasha, R.S., A.M. Khan, and H.V. Manjunatha Reddy. "The Design and EM-Simulation of Square Spiral Inductor Using Simple Equations." *Materials Today: Proceedings* 5, no. 4 (2018): 10875–82. <https://doi.org/10.1016/j.matpr.2018.05.074>.
- [2] Mohan, S.S., M. del Mar Hershenson, S.P. Boyd, and T.H. Lee. "Simple Accurate Expressions for Planar Spiral Inductances." *IEEE Journal of Solid-State Circuits* 34, no. 10 (October 1999): 1419–24. <https://doi.org/10.1109/4.792620>.

See Also

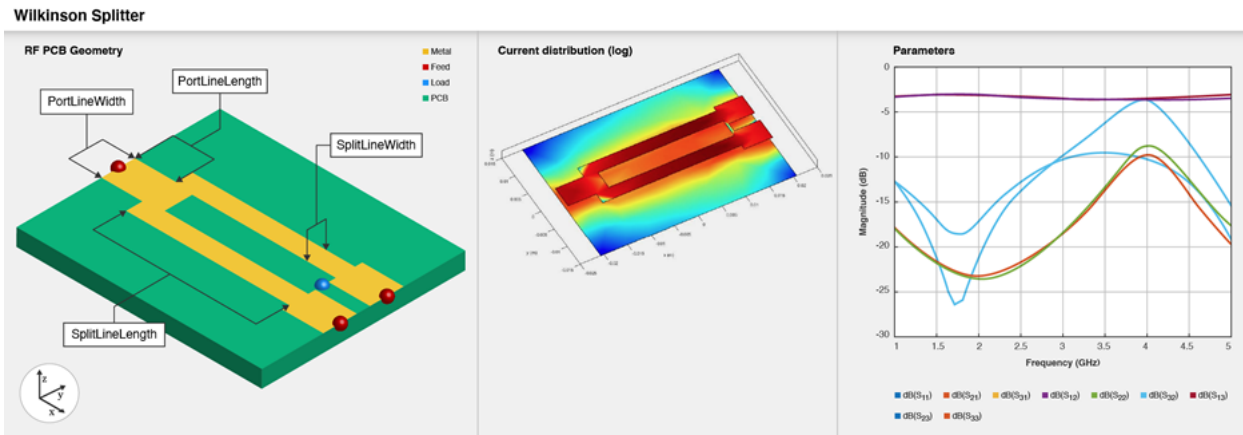
interdigitalCapacitor

wilkinsonSplitter

Create Wilkinson splitter in microstrip form on X-Y plane

Description

Use the `wilkinsonSplitter` object to create a Wilkinson power splitter in microstrip form on the X-Y plane. The Wilkinson power splitter is the most common type of power divider. It is a lossless power divider and provides matching at all ports. The isolation between the output ports is achieved using a $2 \cdot Z_0$ resistor connected between the output ports. The Wilkinson splitter is used in transmitters, receivers, power combining applications, and in devices measuring the power of a test signal.



To analyze the behavioral model for the Wilkinson power splitter, set the Behavioral property in the sparameters to true or 1.

Creation

Syntax

```
splitter = wilkinsonSplitter
splitter = wilkinsonSplitter(Name=Value)
```

Description

`splitter = wilkinsonSplitter` creates a Wilkinson splitter with a Teflon substrate. The default property values are for an operating frequency of 1.8 GHz.

`splitter = wilkinsonSplitter(Name=Value)` sets “Properties” on page 1-28 using one or more name-value arguments. For example, `wilkinsonSplitter(PortLineLength=0.0300)` creates a Wilkinson splitter with an input and output line length of 0.0300 meters. Properties not specified retain their default values.

Properties

Shape — Shape of Wilkinson splitter

"Rectangular" (default) | "Circular"

Shape of the Wilkinson splitter, specified as "Rectangular" or "Circular".

Example: `splitter = wilkinsonSplitter(Shape="Circular")`

Data Types: char | string

PortLineLength — Length of input and output line

0.0060 (default) | positive scalar

Length of the input and the output line in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitter(PortLineLength=0.0070)`

Data Types: double

PortLineWidth — Width of input and output line

0.0049 (default) | positive scalar

Width of the input and the output line in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitter(PortLineWidth=0.0070)`

Data Types: double

SplitLineLength — Length of 70-ohm line

0.0300 (default) | positive scalar

Length of the 70-ohm line in meters, specified as a positive scalar. The typical length of a Wilkinson splitter is $\lambda/4$.

Example: `splitter = wilkinsonSplitter(SplitLineLength=0.0570)`

Data Types: double

SplitLineWidth — Width of 70-ohm line

0.0028 (default) | positive scalar

Width of the 70-ohm line in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitter(SplitLineWidth=0.00780)`

Data Types: double

ResistorLength — Length of resistor

0.0020 (default) | positive scalar

Length of the resistor in meters, specified as a positive scalar. The resistor length determines the distance between the output ports.

Example: `splitter = wilkinsonSplitter(ResistorLength=0.0050)`

Data Types: double

Resistance — Resistance value

100 (default) | positive scalar

Resistance value in ohms, specified as a positive scalar. The default value is for an equal-split Wilkinson splitter.

Example: `splitter = wilkinsonSplitter(Resistance=50)`

Data Types: double

Height — Height of Wilkinson splitter from ground plane

0.0016 (default) | positive scalar

Height of the Wilkinson splitter from the ground plane in meters, specified as a positive scalar. In the case of a multilayer substrate, you can use the Height property to create a Wilkinson splitter where the two dielectrics interface.

Example: `splitter = wilkinsonSplitter(Height=0.0076)`

Data Types: double

GroundPlaneWidth — Width of ground plane in meters

0.0300 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitter(GroundPlaneWidth=0.046)`

Example: double

Substrate — Type of dielectric material

'Teflon' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. The thickness of the default dielectric material Teflon is 0.0016 m or the same value as the Height property.

Example: `d = dielectric("FR4"); splitter = wilkinsonSplitter(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'Copper' (default) | metal object

Type of metal used in the conducting layers, specified as a metal object.

Example: `m = metal("PEC"); splitter = wilkinsonSplitter(Conductor=m)`

Data Types: string | char

Object Functions

charge	Calculate and plot charge distribution
current	Calculate and plot current distribution
design	Design Wilkinson splitter around specified frequency
feedCurrent	Calculate current at feed port
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples

Create Default Wilkinson Splitter

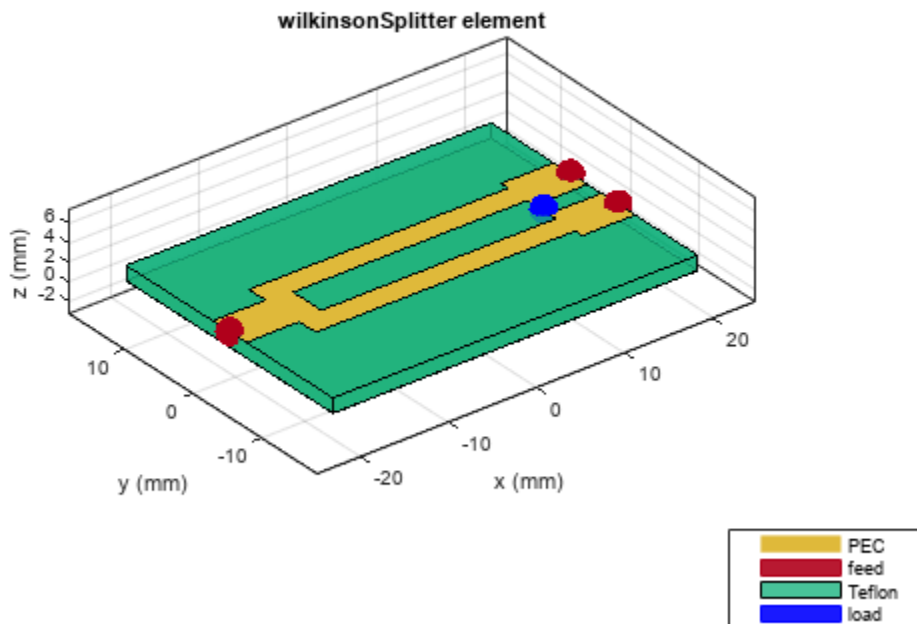
Create and view a default Wilkinson splitter on the X-Y plane.

```
splitter = wilkinsonSplitter

splitter =
  wilkinsonSplitter with properties:

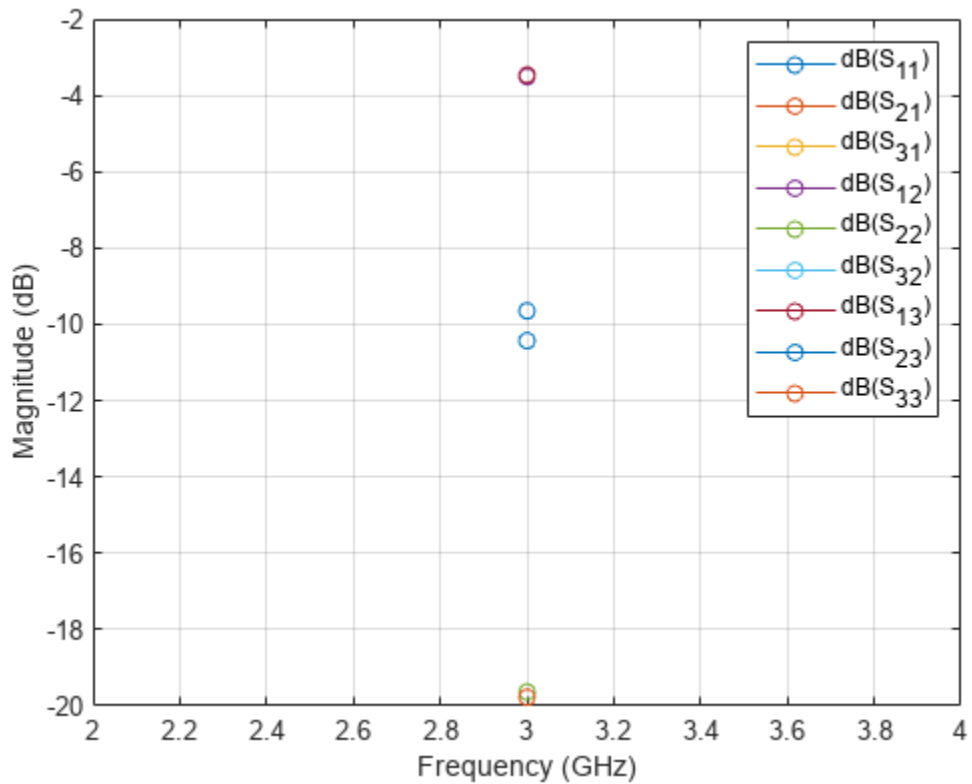
        Shape: 'Rectangular'
    PortLineLength: 0.0060
    PortLineWidth: 0.0049
    SplitLineLength: 0.0300
    SplitLineWidth: 0.0028
    ResistorLength: 0.0020
    Resistance: 100
    Height: 0.0016
    GroundPlaneWidth: 0.0300
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]

show(splitter)
```



Calculate and plot the S-parameters of the splitter at 3 GHz.

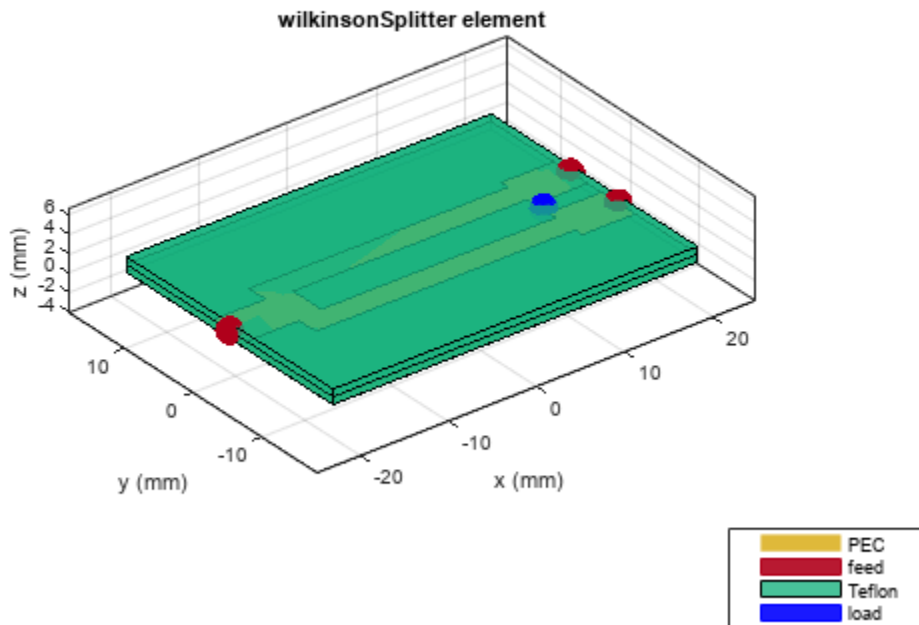
```
spar=sparameters(splitter,3e9);
figure
rfplot(spar);
```



Create Multilayer Wilkinson Splitter

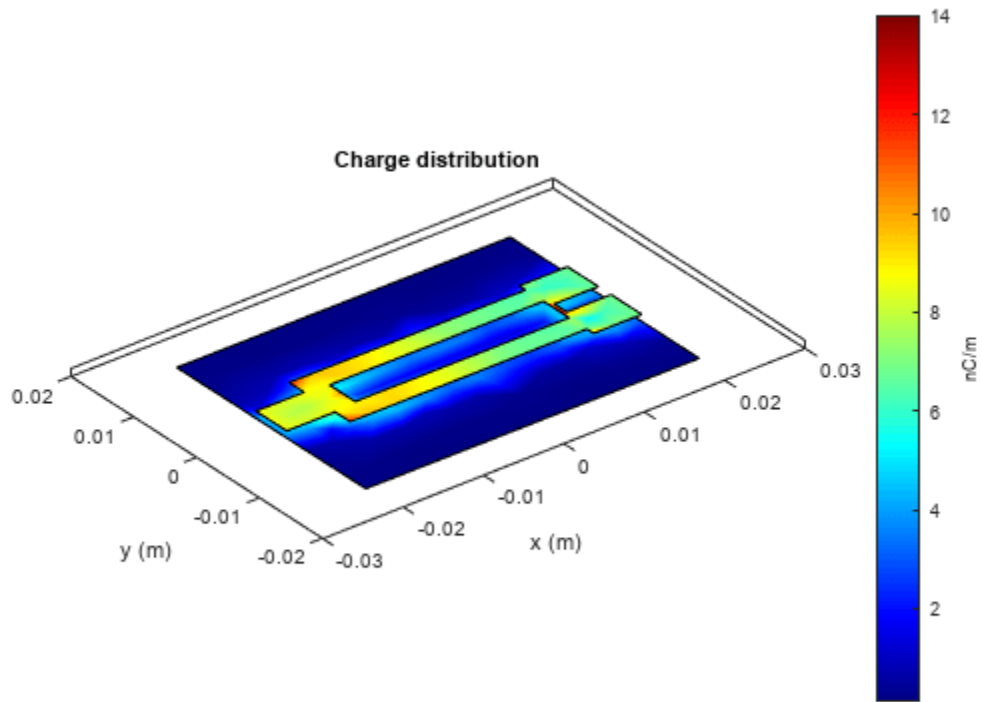
Create and view a multilayer Wilkinson splitter.

```
sub = dielectric(Name=["Teflon","Teflon"],EpsilonR=[2.1 2.1], ...
    LossTangent=[0 0],Thickness=[0.8e-3 0.8e-3]);
splitter = wilkinsonSplitter(Height=0.8e-3,Substrate=sub);
show(splitter)
```

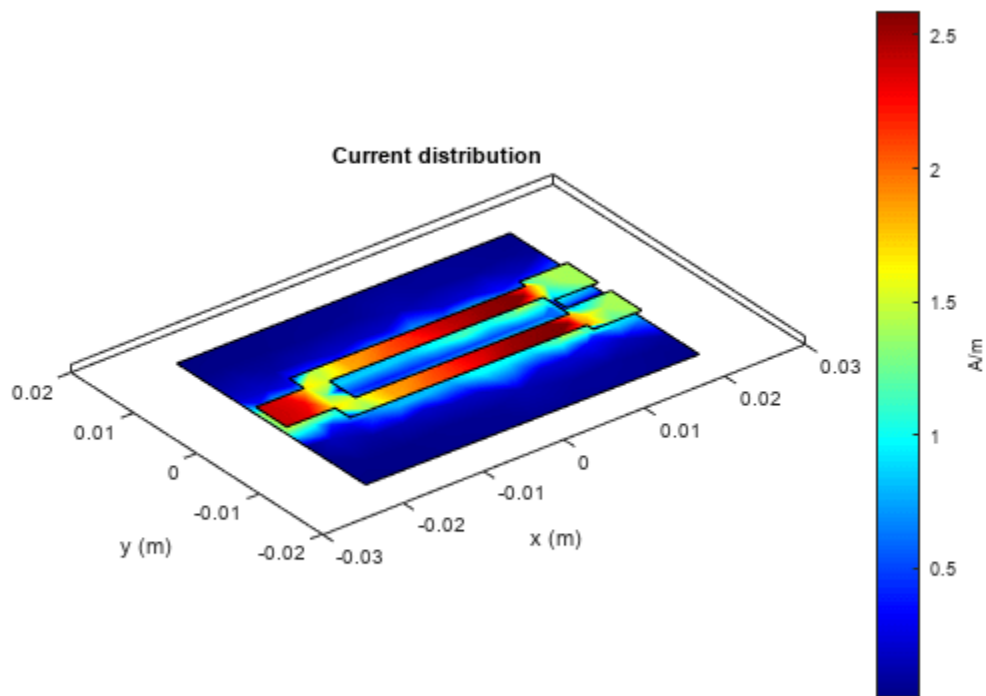


Plot the charge and current on this splitter at 3 GHz.

```
figure  
charge(splitter,3e9)
```



```
figure  
current(splitter,3e9)
```



Version History

Introduced in R2021b

Behavioral Analysis for Wilkinson Power Splitter

Use the `sparameters` function and the `pcbElement` object to perform the behavioral analysis of a Wilkinson power splitter.

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

`wilkinsonSplitterUnequal`

Topics

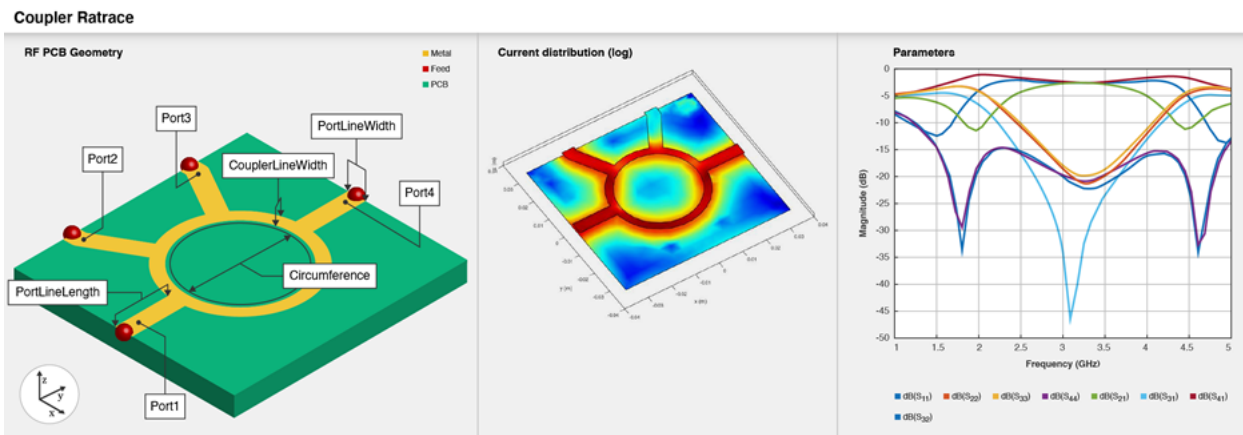
"Behavioral Models"

couplerRatrace

Create equal-split rat-race coupler or 180-degree-ring hybrid

Description

Use the `couplerRatrace` object to create an equal-split rat-race coupler or a 180-degree-ring hybrid.



The rat-race coupler is used as a splitter with a phase shift. When given two inputs, the coupler can create sum and difference ports for added and subtracted power. It also acts as an interface between transmitters and receivers for integrating with an antenna and in building circuits with complex functionality like a comparator with sum and difference ports.

There are four ports and the circumference is 1.5λ . The phase shift between the output ports is 180 degrees. When you apply an input at port 1, port 2 and port 4 are coupled ports, where the output has a phase difference of 180 degrees, and port 3 is the isolated port. When you apply an input at port 3, the output is split equally with same phase at port 2 and port 4.

To analyze the behavioral model for the rat-race coupler, set the `Behavioral` property in the `sparameters` to `true` or `1`.

Creation

Syntax

```
coupler = couplerRatrace
coupler = couplerRatrace(Name=Value)
```

Description

`coupler = couplerRatrace` creates a rat-race coupler. The default property values are for a frequency of 3 GHz.

`coupler = couplerRatrace(Name=Value)` sets “Properties” on page 1-36 using one or more name-value arguments. For example, `couplerRatrace(PortLineLength=0.0286)` creates a rat-race coupler with a port line length of 0.0286 meters. Properties not specified retain their default values.

Properties

PortLineLength — Length of input and output line

0.0186 (default) | positive scalar

Length of the input and the output line in meters, specified as a positive scalar.

Example: `coupler = couplerRatrace(PortLineLength=0.0286)`

Data Types: double

PortLineWidth — Width of input and output line

0.0050 (default) | positive scalar

Width of the input and the output line in meters, specified as a positive scalar.

Example: `coupler = couplerRatrace(PortLineWidth=0.0070)`

Data Types: double

CouplerLineWidth — Width of coupler line

0.0030 (default) | positive scalar

Width of the coupler line in meters, specified as a positive scalar. The default value is for a $\lambda/4$ line with an impedance of $Z_0/\sqrt{2}$ ohms.

Example: `coupler = couplerRatrace(CouplerLineWidth=0.0070)`

Data Types: double

Circumference — Length of coupler line

0.1110 (default) | positive scalar

Length of the coupler line in meters, specified as a positive scalar. The default value is for a 1.5λ line with an impedance of $Z_0/\sqrt{2}$ ohms.

Example: `coupler = couplerRatrace(Circumference=0.2303)`

Data Types: double

Height — Height of rat-race coupler from ground plane

0.0016 (default) | positive scalar

Height of the rat-race coupler from the ground plane in meters, specified as a positive scalar.

In the case of a multilayer substrate, you can use the `Height` property to create a rat-race coupler where the two dielectrics interface.

Example: `coupler = couplerRatrace(Height=0.0015)`

Data Types: double

Substrate — Type of dielectric material

'Teflon' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. The thickness of the default dielectric material Teflon is 0.0016 m or the same as the Height property.

Example: `d = dielectric("FR4"); coupler = couplerRatrace(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'Copper' (default) | metal object

Type of metal used in the conducting layers, specified as a metal object.

Example: `m = metal("PEC"); coupler = couplerRatrace(Conductor=m)`

Data Types: string | char

Object Functions

charge	Calculate and plot charge distribution
coupling	Calculate coupling factor of coupler
current	Calculate and plot current distribution
design	Design rat-race coupler around specified frequency
directivity	Calculate directivity of coupler
feedCurrent	Calculate current at feed port
getZ0	Calculate characteristic impedance of transmission line
isolation	Calculate isolation of coupler
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples

Create Default Rat-Race Coupler

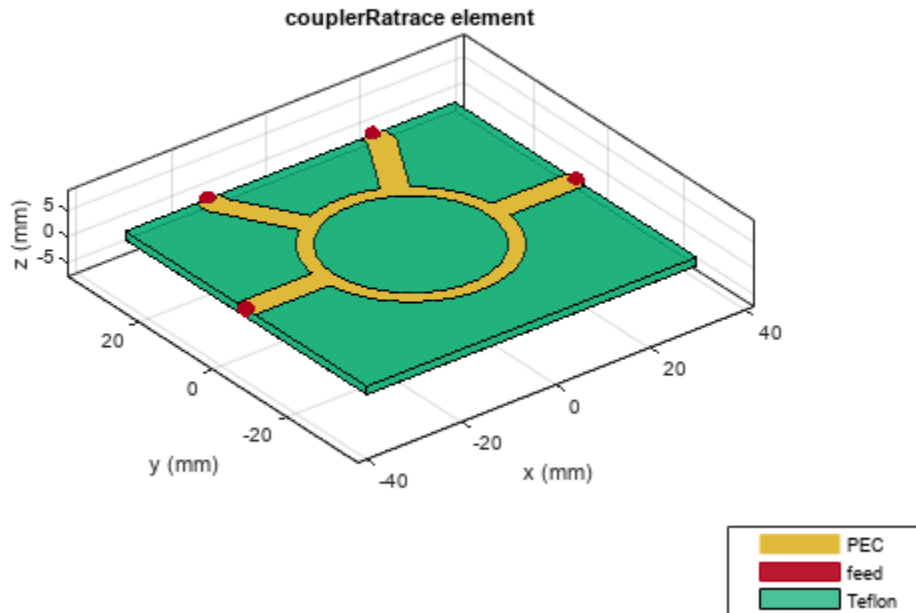
Create and view a default rat-race coupler.

```
ratrace = couplerRatrace
```

```
ratrace =
  couplerRatrace with properties:

    PortLineLength: 0.0186
    PortLineWidth: 0.0050
    CouplerLineWidth: 0.0030
    Circumference: 0.1110
    Height: 0.0016
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

```
show(ratrace)
```



Calculate Current Distribution on Rat-Race Coupler

Create a rat-race coupler with default properties.

```
coupler = couplerRatrace;
```

Set the excitation voltage and the phase angle at the ports of the coupler.

```
v = voltagePort(4)
```

```
v =  
voltagePort with properties:
```

```
    NumPorts: 4  
    FeedVoltage: [1 0 0 0]  
    FeedPhase: [0 0 0 0]  
    PortImpedance: 50
```

```
v.FeedVoltage = [1 0 1 0]
```

```
v =  
voltagePort with properties:
```

```
    NumPorts: 4
```

```

FeedVoltage: [1 0 1 0]
FeedPhase: [0 0 0 0]
PortImpedance: 50

```

```
v.FeedPhase = [90 0 270 0]
```

```

v =
voltagePort with properties:

    NumPorts: 4
    FeedVoltage: [1 0 1 0]
    FeedPhase: [90 0 270 0]
    PortImpedance: 50

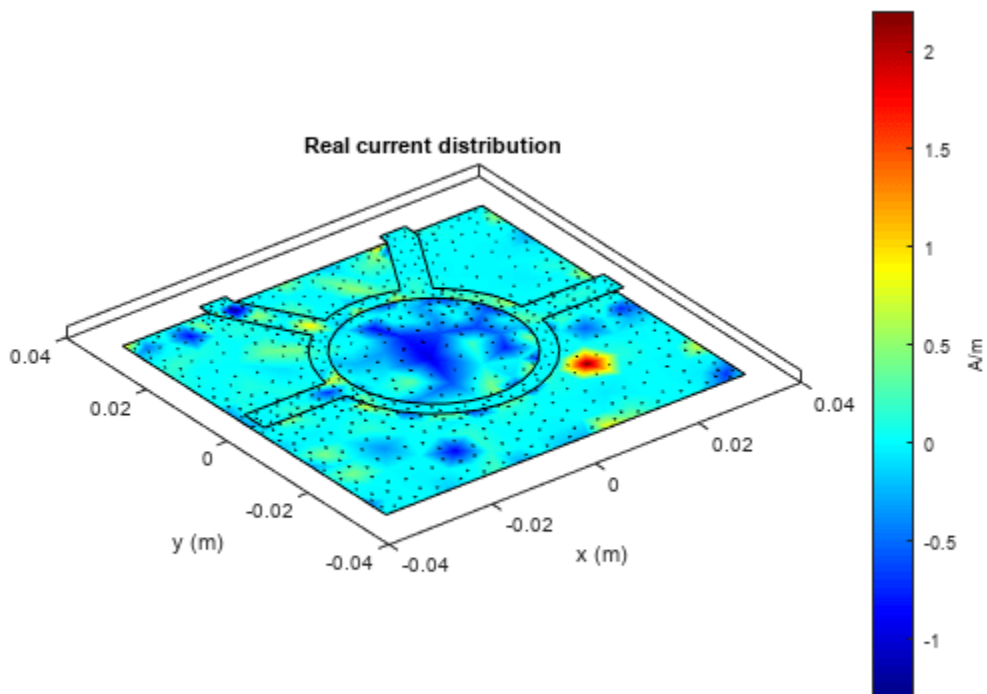
```

Calculate and plot the current on the coupler at 3 GHz.

```

figure
current(coupler,3e9,Excitation=v,Type='real',Direction='on')

```



Version History

Introduced in R2021b

Behavioral Analysis for Rat-Race Coupler

Use the `sparameters` function and the `pcbElement` object to perform the behavioral analysis of a rat-race coupler.

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

`couplerBranchline`

Topics

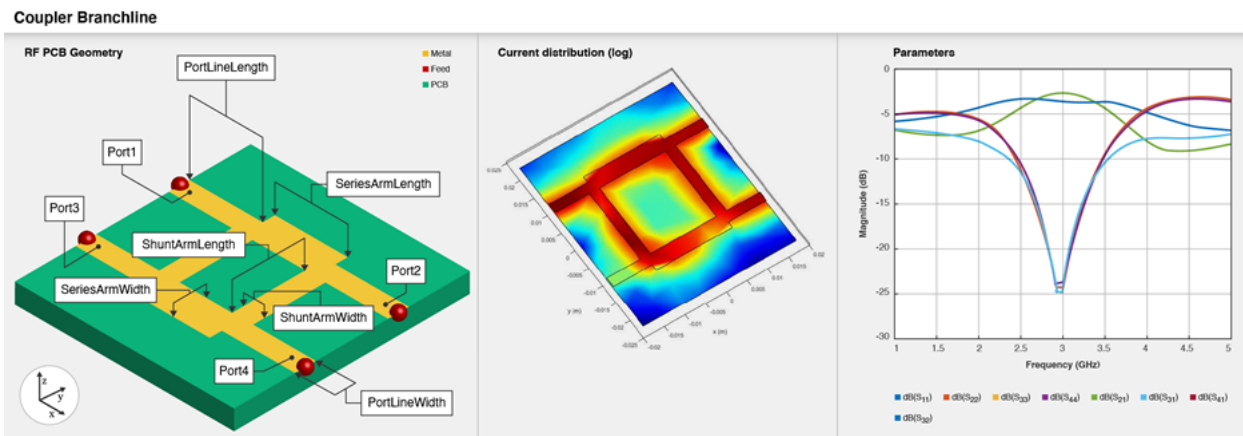
“Behavioral Models”

couplerBranchline

Create branch line coupler or quadrature hybrid

Description

Use the `couplerBranchline` object to create a branch line coupler or a quadrature hybrid. A branch line coupler or a quadrature hybrid divides the power between two ports with a phase difference of 90 degrees. This PCB component has four ports. By default, port 1 is the input port, port 2 is the through port, port 4 is the coupled port, and port 3 is the isolated port.



To analyze the behavioral model for the branchline coupler, set the `Behavioral` property in the `sparameters` to `true` or `1`.

Creation

Syntax

```
coupler = couplerBranchline
coupler = couplerBranchline(Name=Value)
```

Description

`coupler = couplerBranchline` creates a branch line coupler. The default property values are for the frequency of 3 GHz.

`coupler = couplerBranchline(Name=Value)` sets "Properties" on page 1-42 using one or more name-value arguments. For example, `couplerBranchline(PortLineLength=0.0286)` creates a branch line coupler of length 0.0286 meters. Properties not specified retain their default values.

Properties

PortLineLength — Length of input and output line

0.0186 (default) | positive scalar

Length of the input and the output line in meters, specified as a positive scalar.

Example: `coupler = couplerBranchline(PortLineLength=0.0286)`

Data Types: double

PortLineWidth — Width of input and output line

0.0051 (default) | positive scalar

Width of the input and the output line in meters, specified as a positive scalar.

Example: `coupler = couplerBranchline(PortLineWidth=0.0070)`

Data Types: double

SeriesArmLength — Length of series arm

0.0184 (default) | positive scalar

Length of the series arm in meters, specified as a positive scalar.

Example: `coupler = couplerBranchline(SeriesArmLength=0.0286)`

Data Types: double

SeriesArmWidth — Width of series arm

0.0083 (default) | positive scalar

Width of the series arm in meters, specified as a positive scalar.

Example: `coupler = couplerBranchline(SeriesArmWidth=0.0096)`

Data Types: double

ShuntArmLength — Length of shunt arm

0.0186 (default) | positive scalar

Length of the shunt arm in meters, specified as a positive scalar.

Example: `coupler = couplerBranchline(ShuntArmLength=0.0286)`

Data Types: double

ShuntArmWidth — Width of shunt arm

0.0051 (default) | positive scalar

Width of the shunt arm in meters, specified as a positive scalar.

Example: `coupler = couplerBranchline(ShuntArmWidth=0.0096)`

Data Types: double

Height — Height of branch line coupler from ground plane

0.0016 (default) | positive scalar

Height of the branch line coupler from the ground plane in meters, specified as a positive scalar.

In the case of a multilayer substrate, you can use the `Height` property to create a branch line coupler line where the two dielectrics interface.

```
Example: coupler = couplerBranchline(Height=0.0076)
```

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0600 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

```
Example: coupler = couplerBranchline(GroundPlaneWidth=0.046)
```

Example: double

Substrate — Type of dielectric material

'RTDurioid' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a `dielectric` object.

```
Example: d = dielectric("FR4"); coupler = couplerBranchline(Substrate=d)
```

Data Types: string | char

Conductor — Type of metal used in conducting layers

'Copper' (default) | metal object

Type of metal used in the conducting layers, specified as a `metal` object.

```
Example: m = metal("PEC"); coupler = couplerBranchline(Conductor=m)
```

Data Types: string | char

Object Functions

charge	Calculate and plot charge distribution
coupling	Calculate coupling factor of coupler
current	Calculate and plot current distribution
design	Design branchline coupler around particular frequency
directivity	Calculate directivity of coupler
feedCurrent	Calculate current at feed port
getZ0	Calculate characteristic impedance of transmission line
isolation	Calculate isolation of coupler
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples

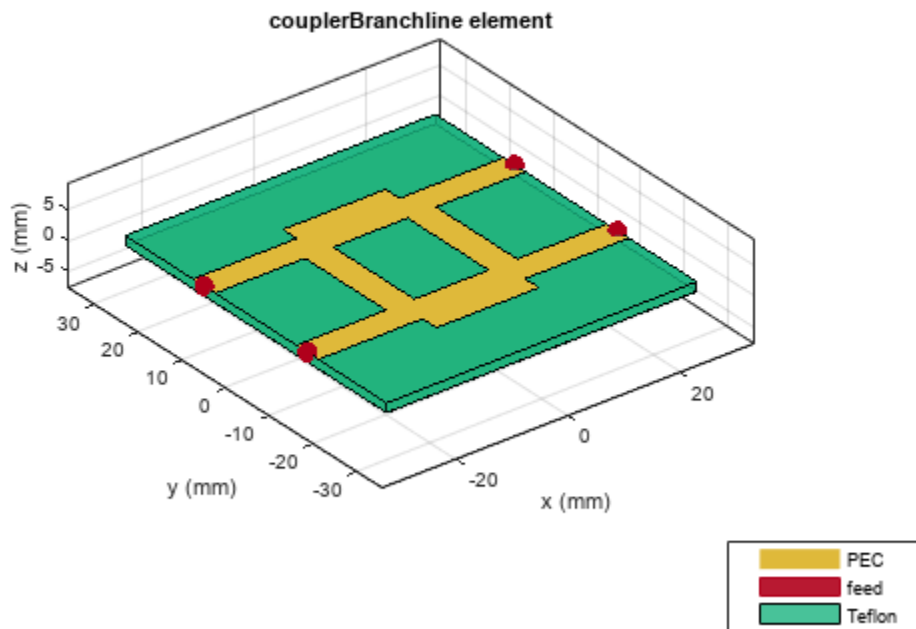
Create Default Branchline Coupler

Create and view a default branchline coupler.

```
coupler = couplerBranchline
```

```
coupler =  
  couplerBranchline with properties:  
  
    PortLineLength: 0.0186  
    PortLineWidth: 0.0051  
    SeriesArmLength: 0.0184  
    SeriesArmWidth: 0.0083  
    ShuntArmLength: 0.0186  
    ShuntArmWidth: 0.0051  
    Height: 0.0016  
    GroundPlaneWidth: 0.0600  
    Substrate: [1x1 dielectric]  
    Conductor: [1x1 metal]
```

```
show(coupler)
```



Version History

Introduced in R2021b

Behavioral Analysis for Branchline Coupler

Use the `sparameters` function and the `pcbElement` object to perform the behavioral analysis of a branchline coupler.

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

couplerRatrace

Topics

“Behavioral Models”

interdigitalCapacitor

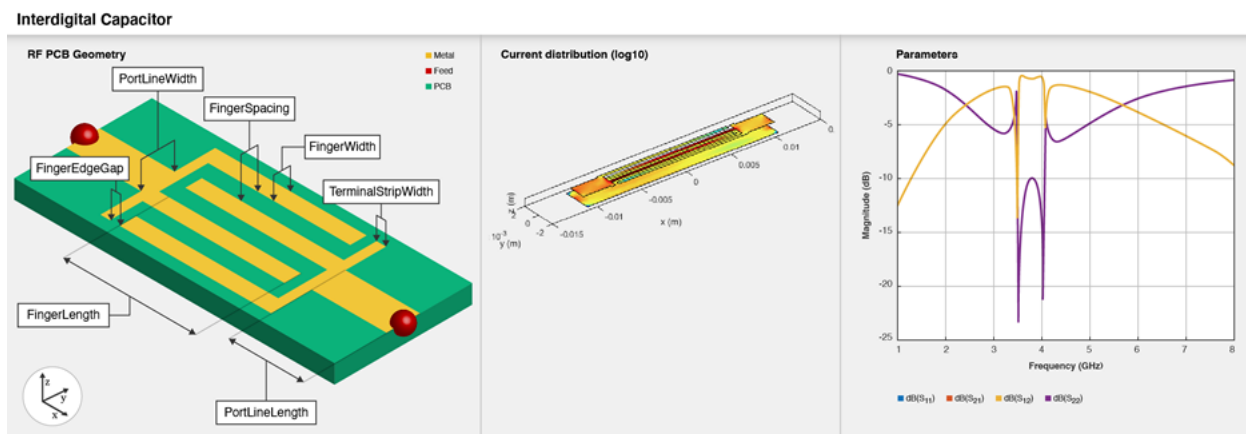
Create basic interdigital capacitor

Description

Use the `interdigitalCapacitor` object to create an interdigital planar capacitor (IDC). IDCs are used in high frequency applications such as:

- Receiver circuits where antenna radiators are connected to RF
- Wireless data communications with RFID
- Humidity and solution concentration measurements
- Lab-on-chip devices (LOCs)

Note This PCB object supports behavioral modeling. For more information, see “Behavioral Models”. To analyze the behavioral model for a interdigital capacitor, set the `Behavioral` property in the `sparameters` function to `true` or `1`



A two-port series IDC with microstrip form feeder lines supports single and multiple dielectrics. It is a coplanar structure consisting of multiple comb electrodes or intersecting fingers with spaces between the fingers. An IDC can have identical port line lengths and widths on either sides.

Creation

Syntax

```
capacitor = interdigitalCapacitor
capacitor = interdigitalCapacitor(Name=Value)
```

Description

`capacitor = interdigitalCapacitor` creates a basic interdigital capacitor. The default property values are for an operating bandwidth of 3.6-4 GHz.

`capacitor = interdigitalCapacitor(Name=Value)` sets “Properties” on page 1-47 using one or more name-value arguments. For example, `interdigitalCapacitor(NumFingers=10)` creates an interdigital capacitor with 10 fingers. Properties not specified retain their default values.

Properties

NumFingers — Number of fingers on capacitor

4 (default) | positive scalar

Number of fingers on the capacitor, specified as a positive scalar.

Example: `capacitor = interdigitalCapacitor(NumFingers=10)`

Data Types: double

FingerLength — Length of overlapping fingers

0.0137 (default) | positive scalar

Length of the overlapping fingers in meters, specified as a positive scalar.

Example: `capacitor = interdigitalCapacitor(FingerLength=0.0217)`

Data Types: double

FingerWidth — Width of overlapping fingers

3.1600e-04 (default) | positive scalar

Width of the overlapping fingers in meters, specified as a positive scalar.

Example: `capacitor = interdigitalCapacitor(FingerWidth=4.8000e-04)`

Data Types: double

FingerSpacing — Distance between fingers

3.0000e-04 (default) | positive scalar

Distance between the fingers in meters, specified as a positive scalar.

Example: `capacitor = interdigitalCapacitor(FingerSpacing=2.9000e-04)`

Data Types: double

FingerEdgeGap — Gap between edges of fingers

3.4100e-04 (default) | positive scalar

Gap between the edges of the fingers in meters, specified as a positive scalar.

Example: `capacitor = interdigitalCapacitor(FingerEdgeGap=2.05000e-04)`

Data Types: double

TerminalStripWidth — Width of terminals

5.0000e-04 (default) | positive scalar

Width of the terminals in meters, specified as a positive scalar.

Example: `capacitor = interdigitalCapacitor(TerminalStripWidth=4.9000e-04)`

Data Types: double

PortLineWidth — Width of ports

0.0019 (default) | positive scalar

Width of the ports in meters, specified as a positive scalar.

Example: `capacitor = interdigitalCapacitor(PortLineWidth=0.0020)`

Data Types: double

PortLineLength — Length of ports

0.0030 (default) | positive scalar

Length of the ports in meters, specified as a positive scalar.

Example: `capacitor = interdigitalCapacitor(PortLineLength=0.0040)`

Data Types: double

Height — Height from ground plane to capacitor

7.8700e-04 (default) | positive scalar

Height from the capacitor to the ground plane in meters, specified as a positive scalar.

Example: `capacitor = interdigitalCapacitor(Height=6.9000e-04)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0030 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `capacitor = interdigitalCapacitor(GroundPlaneWidth=0.0040)`

Example: double

Substrate — Type of dielectric material

dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. The default value is a dielectric object with these properties:

- Name—{'Roger'}
- EpsilonR—3.2
- LossTangent—0.0002
- Thickness—0.000787

Example: `d = dielectric("FR4"); capacitor = interdigitalCapacitor(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'Copper' (default) | metal object

Type of metal used in the conducting layers, specified as a `metal` object.

Example: `m = metal("PEC"); capacitor =interdigitalCapacitor(Conductor=m)`

Data Types: `string` | `char`

Object Functions

<code>capacitance</code>	Calculate capacitance
<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>feedCurrent</code>	Calculate current at feed port
<code>getZ0</code>	Calculate characteristic impedance of transmission line
<code>layout</code>	Plot all metal layers and board shape
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>shapes</code>	Extract all metal layer shapes of PCB component
<code>show</code>	Display PCB component structure or PCB shape
<code>sparameters</code>	Calculate S-parameters for RF PCB objects

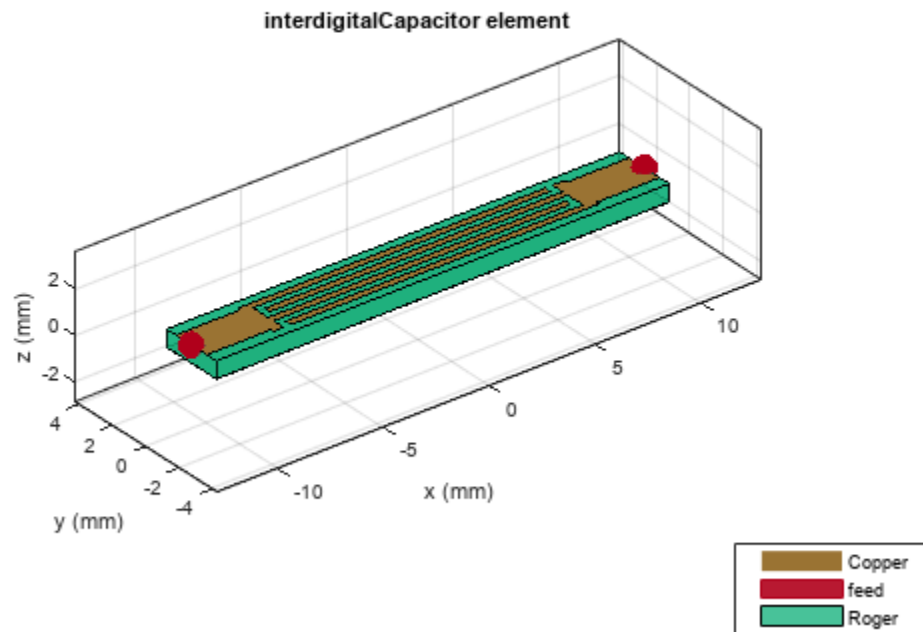
Examples

Create Default Interdigital Capacitor

Create and view a default interdigital capacitor.

```
idcapacitor = interdigitalCapacitor
idcapacitor =
    interdigitalCapacitor with properties:
        NumFingers: 4
        FingerLength: 0.0137
        FingerWidth: 3.1600e-04
        FingerSpacing: 3.0000e-04
        FingerEdgeGap: 3.4100e-04
        TerminalStripWidth: 5.0000e-04
        PortLineWidth: 0.0019
        PortLineLength: 0.0030
        Height: 7.8700e-04
        GroundPlaneWidth: 0.0030
        Substrate: [1x1 dielectric]
        Conductor: [1x1 metal]

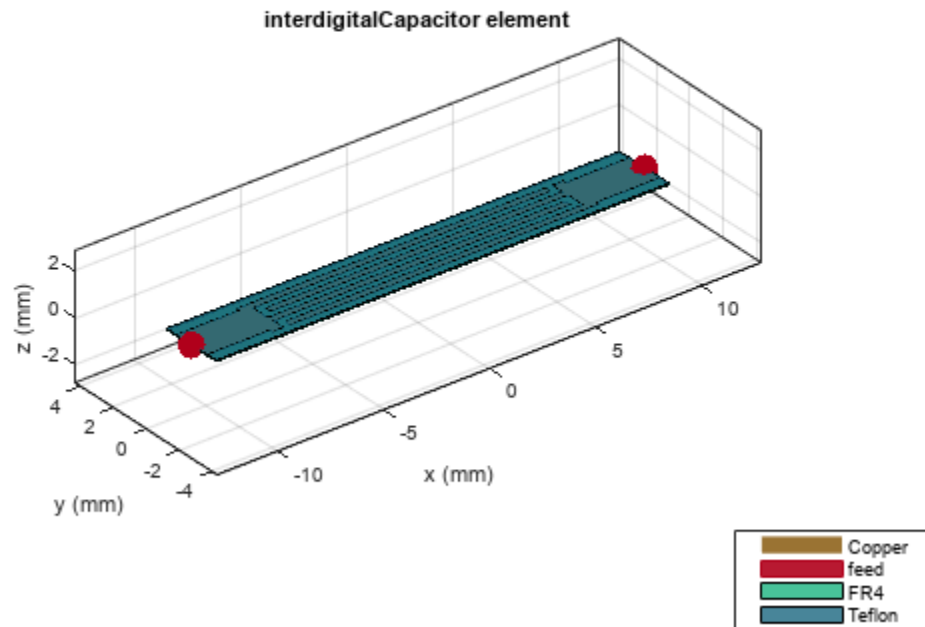
show(idcapacitor)
```



Multilayer Interdigital Capacitor

Create and view a multilayer interdigital capacitor with two different dielectrics.

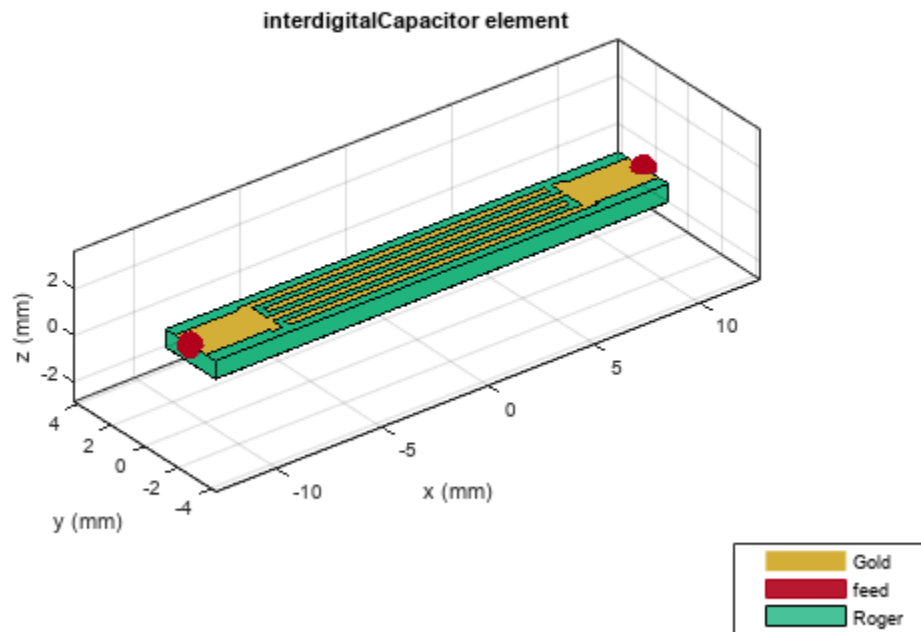
```
idcapacitor = interdigitalCapacitor;  
sub = dielectric("FR4", "Teflon");  
sub.Thickness = [0.00003 0.00003];  
idcapacitor.Substrate = sub;  
idcapacitor.Height = 0.00003;  
show(idcapacitor);
```



Analyze Interdigital Capacitor Using Behavioral S-parameters

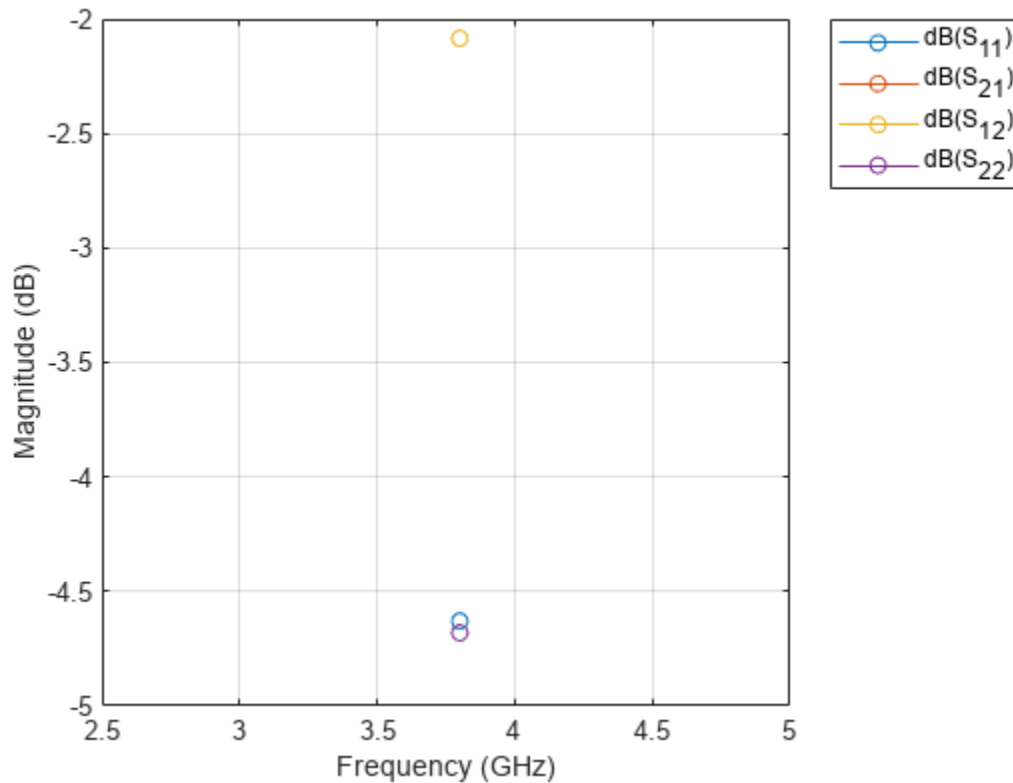
Create an interdigital capacitor using gold as the conductor.

```
capacitor = interdigitalCapacitor;  
capacitor.Conductor = metal("Gold");  
show(capacitor)
```



Compute and plot the behavioral S-parameters of the capacitor at 3.8 GHz.

```
spar = sparameters(capacitor,3.8e9,Behavioral=true);  
rfplot(spar)
```

More About

Parametric Analysis Guidelines

- The capacitance of an interdigital capacitor is directly proportional to its physical parameters such as NumFingers, FingerLength, FingerWidth, and FingerSpacing.
- Increasing the height of the dielectric decreases the parasitic capacitance.

To design the capacitor at high frequency consider the following assumptions:

- Increasing the number of fingers creates a periodic and smooth structure.
- Capacitor dimensions should be much smaller than the quarter wavelength.

Version History

Introduced in R2021b

References

- [1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.
- [2] Jungreuthmayer, Christian, Gerald M. Birnbaumer, Peter Ertl, and Jürgen Zanghellini. "Improving the Measurement Sensitivity of Interdigital Dielectric Capacitors (IDC) by Optimizing the

- Dielectric Property of the Homogeneous Passivation Layer." *Sensors and Actuators B: Chemical* 162, no. 1 (February 2012): 418-24. <https://doi.org/10.1016/j.snb.2011.12.009>.
- [3] Ruppin, R. "Surface Polaritons of a Left-Handed Material Slab." *Journal of Physics: Condensed Matter* 13, no. 9 (March 5, 2001): 1811-18. <https://doi.org/10.1088/0953-8984/13/9/304>.
- [4] Caloz, Christophe, and Tatsuo Itoh. *Electromagnetic Metamaterials: Transmission Line Theory and Microwave Applications: The Engineering Approach*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2005. <https://doi.org/10.1002/0471754323>.

See Also

spiralInductor

bendCurved

Create curved bend shape on X-Y plane

Description

Use the bendCurved object to create a curved bend shape on the X-Y plane.

Note This shape object supports behavioral modeling. For more information, see “Behavioral Models”.

Creation

Syntax

```
bend = bendCurved
bend = bendCurved(Name=Value)
```

Description

bend = bendCurved creates a curved bend shape on the X-Y plane.

bend = bendCurved(Name=Value) sets “Properties” on page 1-55 using one or more name-value arguments. For example, bendCurved(ReferencePoint=[1 1]) creates a curved bend shape with the reference point at [1 1]. Properties not specified retain their default values.

Properties

Name — Name of curved bend shape

'myCurvedbend' (default) | character vector | string scalar

Name of the curved bend shape, specified as a character vector or a string scalar.

Example: bend = bendCurved(Name="bendcurve1")

Data Types: char

ReferencePoint — Reference point

[0 0] (default) | two-element vector

Reference point for the curved bend shape in Cartesian coordinates, specified as a two-element vector.

Example: bend = bendCurved(ReferencePoint=[1 1])

Data Types: double

Length — Length of curved bend shape

[0.0100 0.01000] (default) | two-element vector

Length of the curved bend shape in meters, specified as a two-element vector.

Example: `bend = bendCurved(Length=[0.0500 0.0500])`

Data Types: double

Width — Width of curved bend shape

`[0.0500 0.0500]` (default) | two-element vector

Width of the curved bend shape in meters, specified as a two-element vector.

Example: `bend = bendCurved(Width=[0.0100 0.0100])`

Data Types: double

CurveRadius — Radius of corner

`0.0035` (default) | positive scalar

Radius of the corner in meters, specified as a positive scalar.

Example: `bend = bendCurved(CurveRadius=2)`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples

Create Default Curved Bend Shape

Create a curved bend shape with default properties.

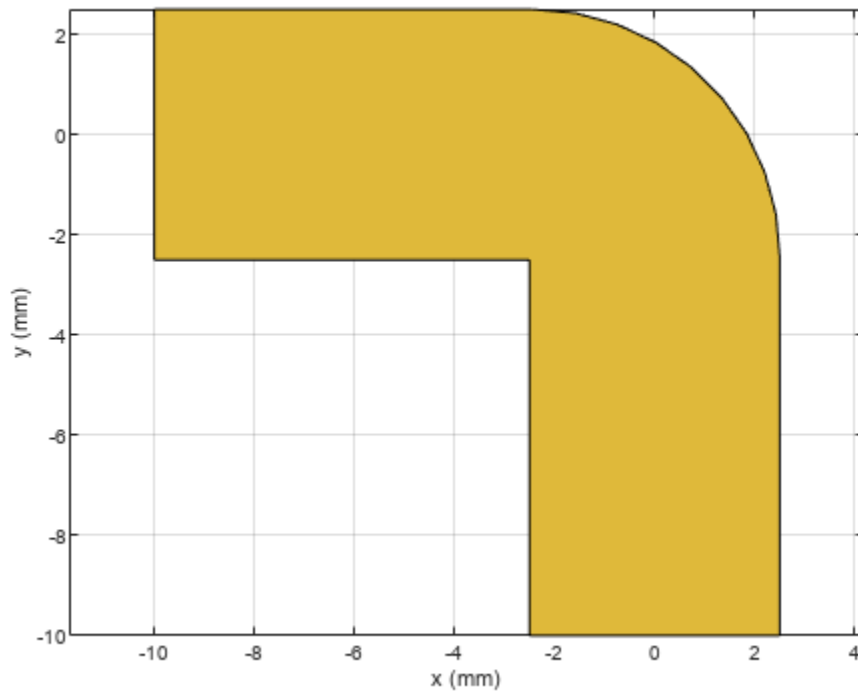
```
bend = bendCurved
```

```
bend =  
    bendCurved with properties:  
        Name: 'myCurvedbend'  
    ReferencePoint: [0 0]  
        Length: [0.0100 0.0100]
```

```
Width: [0.0050 0.0050]
CurveRadius: 0.0035
```

View the shape.

```
show(bend)
```



Mesh Rotated Curved Bend Shape

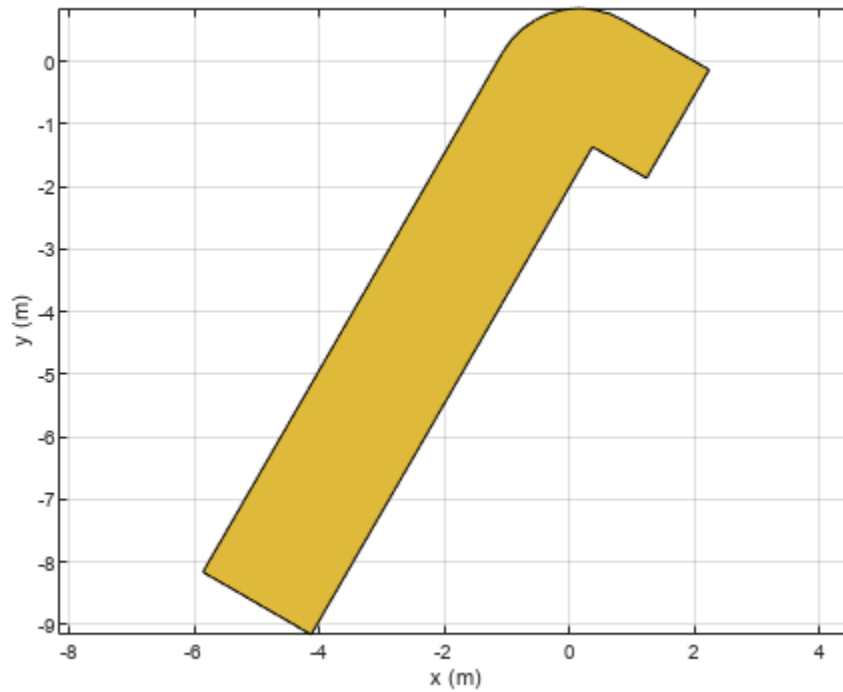
Create a curved bend shape of lengths of 10 m and 2 m, width of 2 m, and rotate it about the Z-axis by 60 degrees.

```
bend = bendCurved(Length=[10 2],Width=[2 2],CurveRadius=1)
```

```
bend =
  bendCurved with properties:
```

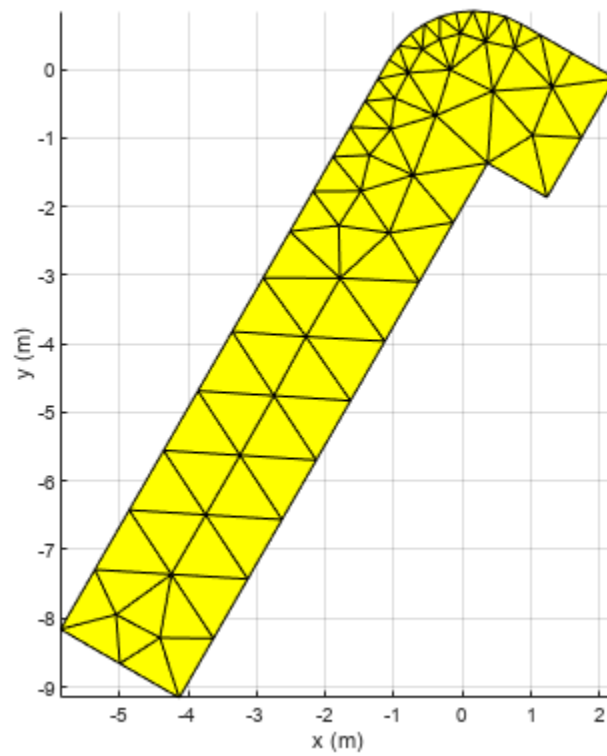
```
    Name: 'myCurvedbend'
ReferencePoint: [0 0]
    Length: [10 2]
    Width: [2 2]
    CurveRadius: 1
```

```
bend = rotateZ(bend,60);  
show(bend)
```



Mesh the curved bend shape at a maximum edge length of 1 m.

```
meshconfig(bend,"manual")  
  
ans = struct with fields:  
    NumTriangles: 0  
    NumTetrahedra: 0  
    NumBasis: []  
    MaxEdgeLength: []  
    MinEdgeLength: []  
    GrowthRate: []  
    MeshMode: 'manual'  
  
mesh(bend,MaxEdgeLength=1)
```



Version History

Introduced in R2021b

See Also

`bendRightAngle` | `bendMitered`

bendMitered

Create mitered bend shape on X-Y plane

Description

Use the `bendMitered` object to create a mitered bend shape on the X-Y plane.

Note This shape object supports behavioral modeling. For more information, see “Behavioral Models”.

Creation

Syntax

```
bend = bendMitered  
bend = bendMitered(Name=Value)
```

Description

`bend = bendMitered` creates a mitered bend shape on the X-Y plane.

`bend = bendMitered(Name=Value)` sets “Properties” on page 1-60 using one or more name-value arguments. For example, `bendMitered(ReferencePoint=[1 1])` creates a mitered bend shape with the reference point at [1 1]. Properties not specified retain their default values.

Properties

Name — Name of mitered bend shape

'myMiteredbend' (default) | character vector | string scalar

Name of the mitered bend shape, specified as a character vector or a string scalar.

Example: `bend = bendMitered(Name="bendmitered1")`

Data Types: char

ReferencePoint — Reference point

[0 0] (default) | two-element vector

Reference point for the mitered bend shape in Cartesian coordinates, specified as a two-element vector.

Example: `bend = bendMitered(ReferencePoint=[1 2])`

Data Types: double

Length — Length of mitered bend shape

[0.0100 0.0100] (default) | two-element vector

Length of the mitered bend shape in meters, specified as a two-element vector.

Example: `bend = bendMitered(Length=[0.005 0.005])`

Data Types: double

Width — Width of mitered bend shape

`[0.0050 0.0500]` (default) | two-element vector

Width of the mitered bend shape in meters, specified as a two-element vector.

Example: `bend = bendMitered(Width=[1 1])`

Data Types: double

MiterDiagonal — Length of miter diagonal

`0.0035` (default) | positive scalar

Length of the miter diagonal in meters, specified as a positive scalar.

Example: `bend = bendMitered(MiterDiagonal=2)`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples

Create Default Mitered Bend

Create a mitered bend with default properties.

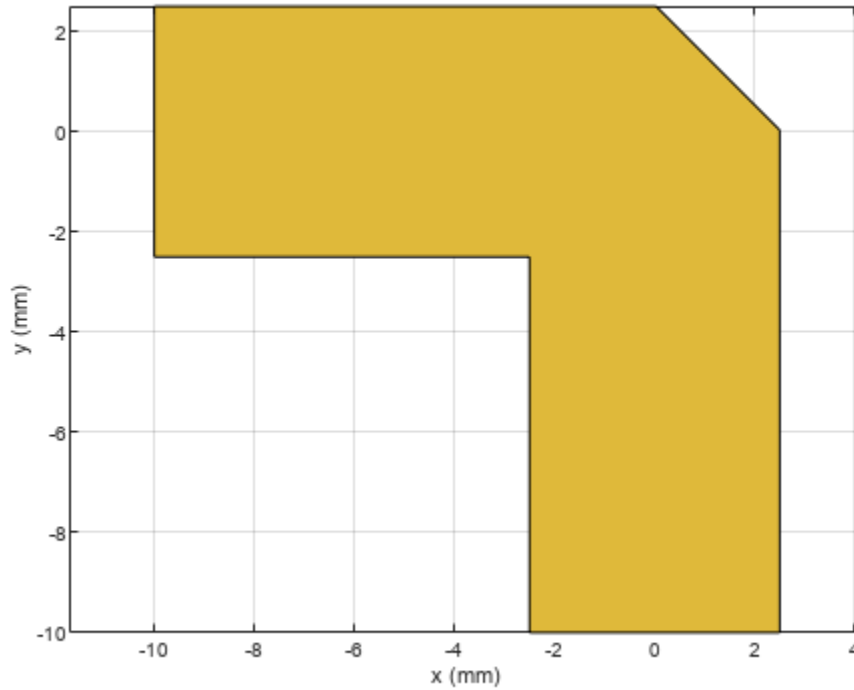
```
bend = bendMitered
```

```
bend =
  bendMitered with properties:
      Name: 'myMiteredbend'
  ReferencePoint: [0 0]
      Length: [0.0100 0.0100]
```

```
Width: [0.0050 0.0050]  
MiterDiagonal: 0.0035
```

View the shape.

```
show(bend)
```



Mesh Rotated Mitered Bend Shape

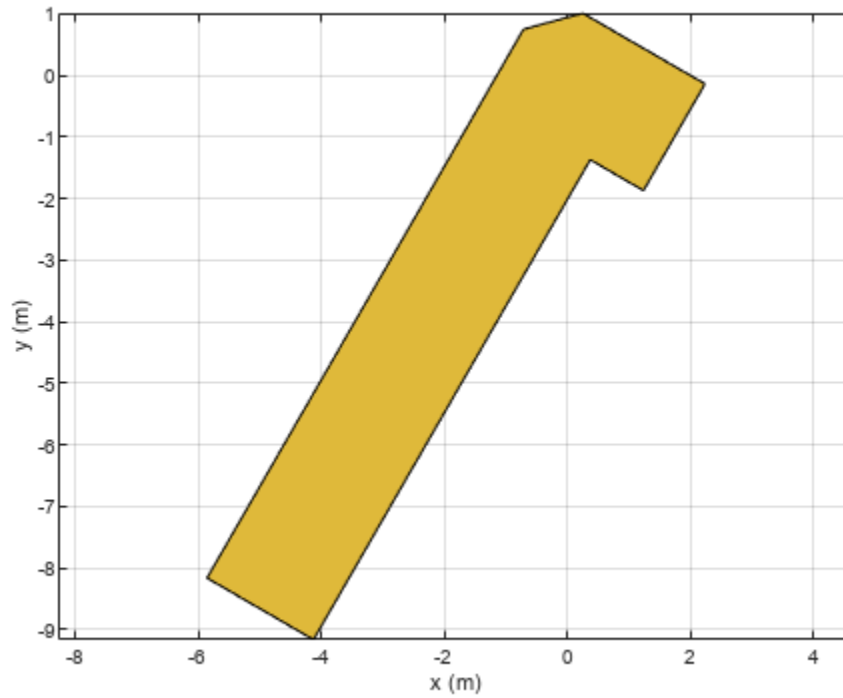
Create a mitered bend shape of lengths of 10 m and 2 m, width of 2 m, and rotate it about the Z-axis by 60 degrees.

```
bend = bendMitered(Length=[10 2],Width=[2 2],MiterDiagonal=1);  
bend = rotateZ(bend,60)
```

```
bend =  
  bendMitered with properties:
```

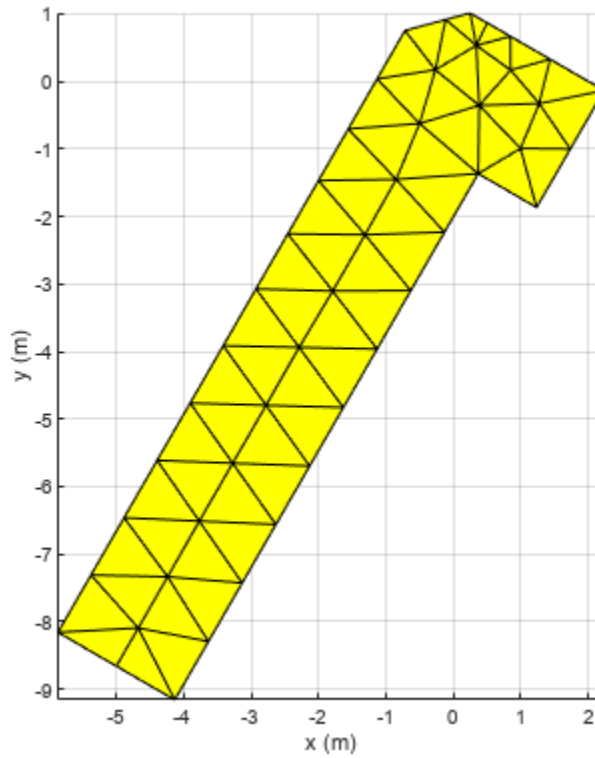
```
      Name: 'myMiteredbend'  
ReferencePoint: [0 0]  
      Length: [10 2]  
      Width: [2 2]  
MiterDiagonal: 1
```

```
show(bend)
```



Mesh the mitered bend shape at a maximum edge length of 1 m.

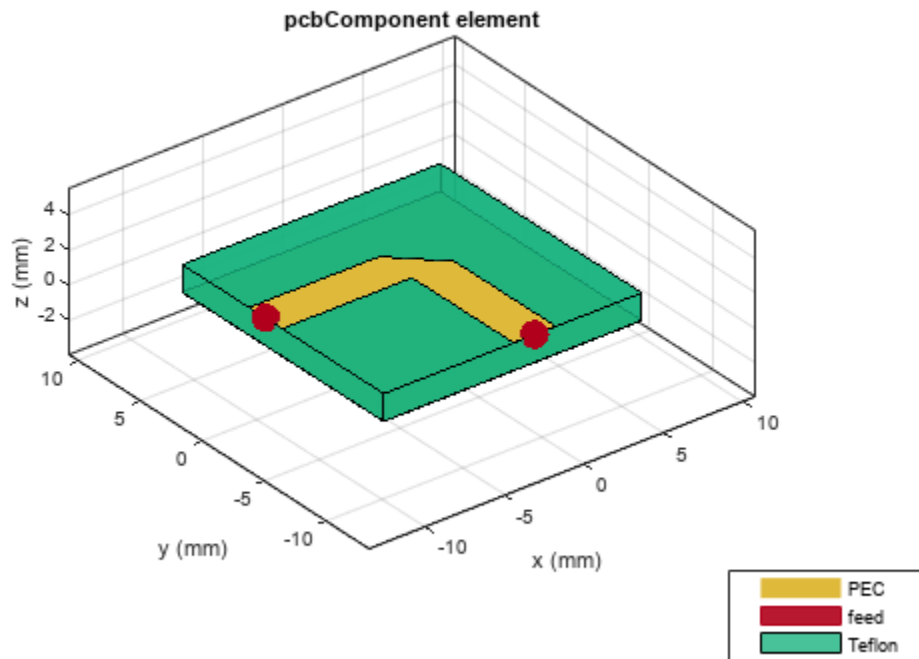
```
mesh(bend,MaxEdgeLength=1)
```



Use Behavioral Model to Calculate S-Parameters of Mitered Bend Microstrip

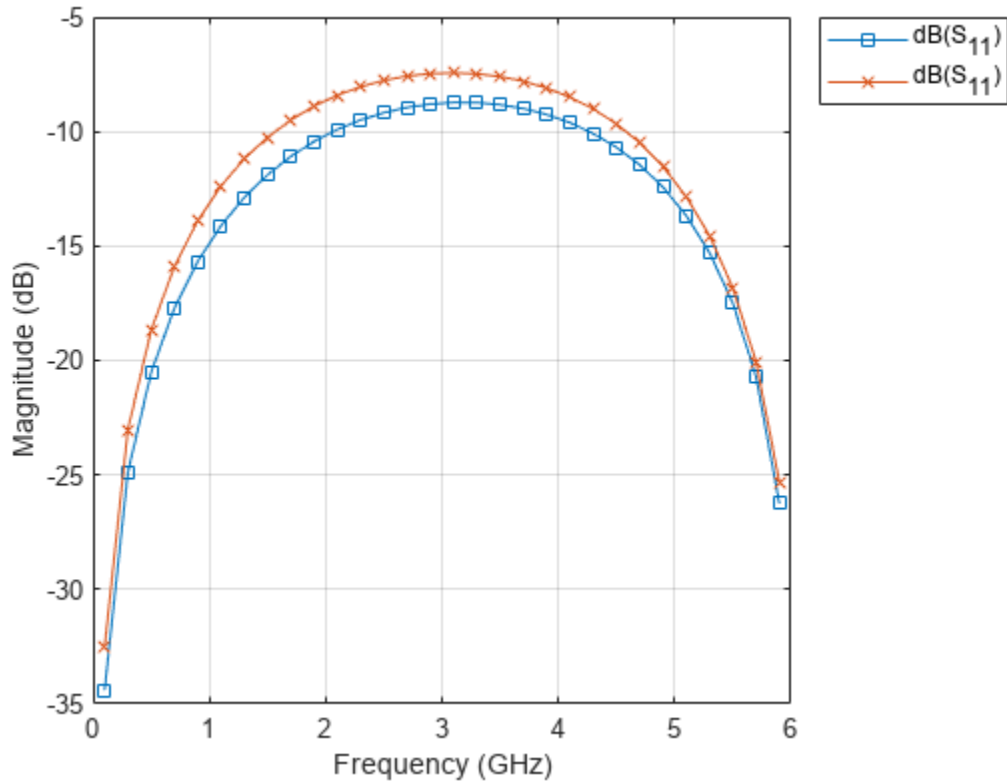
Create mitered bend microstrip.

```
m = design(microstripLine,6e9,"Z0",75);
layer2d = bendMitered('Length',[m.Length/2 m.Length/2],...
'Width',[m.Width m.Width],'MiterDiagonal',sqrt(2)*m.Width);
robj = pcbComponent(layer2d);
robj.BoardThickness = m.Substrate.Thickness;
robj.Layers{2} = m.Substrate;
show(robj)
```



Compute and plot s-parameters.

```
freq = (1:2:60)*100e6;  
Sckt = sparameters(robject, freq, 'Behavioral', true);  
Sem = sparameters(robject, freq);  
rfplot(Sckt, 1, 1, 'db', '-s')  
hold on  
rfplot(Sem, 1, 1, 'db', '-x')
```



Reference:

M. Kirschning, R. H. Jansen and N. H. L. Koster, "Measurement and Computer-Aided Modeling of Microstrip Discontinuities by an Improved Resonator Method," 1983 IEEE MTT-S International Microwave Symposium Digest, Boston, MA, USA, 1983, pp. 495-497, doi: 10.1109/MWSYM.1983.1130959.

Version History

Introduced in R2021b

See Also

`bendRightAngle` | `bendCurved`

bendRightAngle

Create right-angle bend shape on X-Y plane

Description

Use the `bendRightAngle` object to create a right-angle bend shape on the X-Y plane.

Note This shape object supports behavioral modeling. For more information, see “Behavioral Models”.

Creation

Syntax

```
bend = bendRightAngle  
bend = bendRightAngle(Name=Value)
```

Description

`bend = bendRightAngle` creates a right-angle bend shape on the X-Y plane.

`bend = bendRightAngle(Name=Value)` sets “Properties” on page 1-67 using one or more name-value arguments. For example, `bendRightAngle(ReferencePoint=[1 1])` creates a right-angle bend shape with the reference point at [1 1]. Properties not specified retain their default values.

Properties

Name — Name of right-angle bend shape

'myRightAnglebend' (default) | character vector | string scalar

Name of the right-angle bend shape, specified as a character vector or a string scalar.

Example: `bend = bendRightAngle(Name="bendrightangle1")`

Data Types: char

ReferencePoint — Reference point

[0 0] (default) | two-element vector

Reference point for the right-angle bend shape in Cartesian coordinates, specified as a two-element vector.

Example: `bend = bendRightAngle(ReferencePoint=[1 2])`

Data Types: double

Length — Length of right-angle bend shape

[0.0100 0.0100] (default) | two-element vector

Length of the right-angle bend shape in meters, specified as a two-element vector.

Example: `bend = bendRightAngle(Length=[0.0500 0.0500])`

Data Types: double

Width — Width of right-angle bend shape

`[0.0050 0.0050]` (default) | two-element vector

Width of the right-angle bend shape in meters, specified as a two-element vector.

Example: `bend = bendRightAngle(Width=[0.0200 0.0200])`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples

Create Default Right-Angle Bend

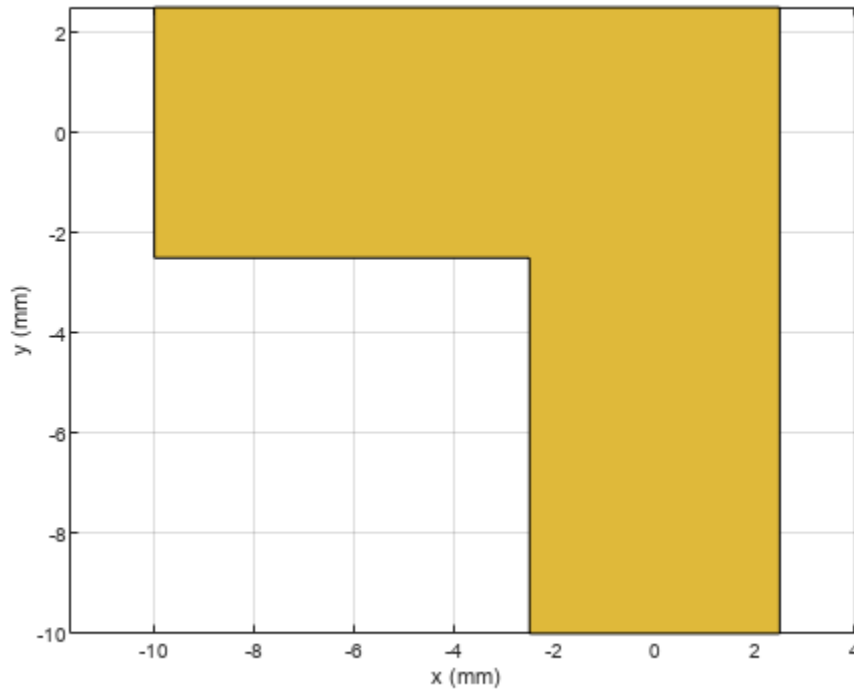
Create a right-angle bend with default properties.

```
bend = bendRightAngle
```

```
bend =  
    bendRightAngle with properties:  
        Name: 'myRightAnglebend'  
        ReferencePoint: [0 0]  
        Length: [0.0100 0.0100]  
        Width: [0.0050 0.0050]
```

View the shape.

```
show(bend)
```

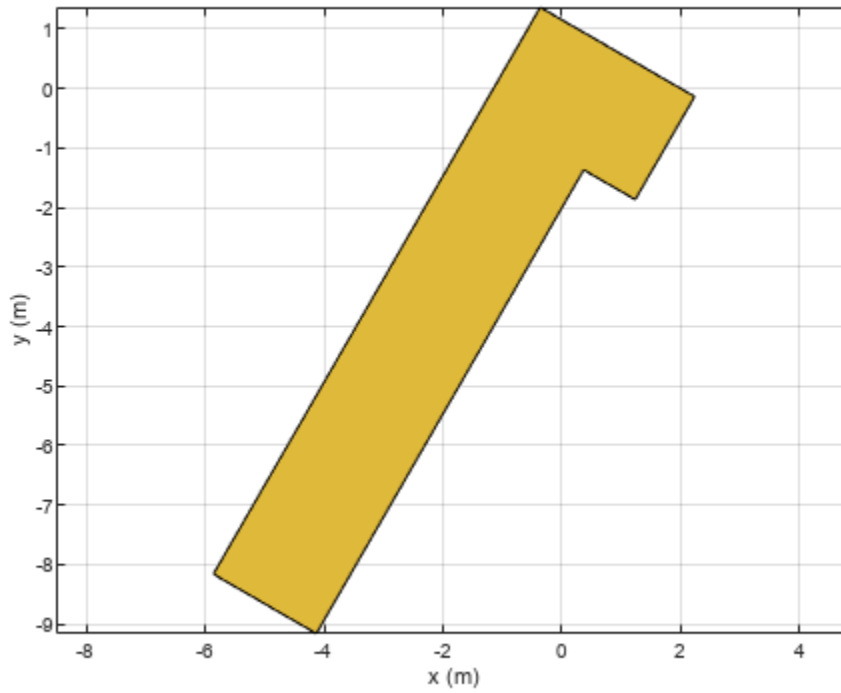
Mesh Rotated Right-Angle Bend Shape

Create a right-angle bend shape of lengths of 10 m and 2 m, width of 2m, and rotate it about the Z-axis by 60 degrees.

```
bend = bendRightAngle(Length=[10 2],Width=[2 2]);  
bend = rotateZ(bend,60)
```

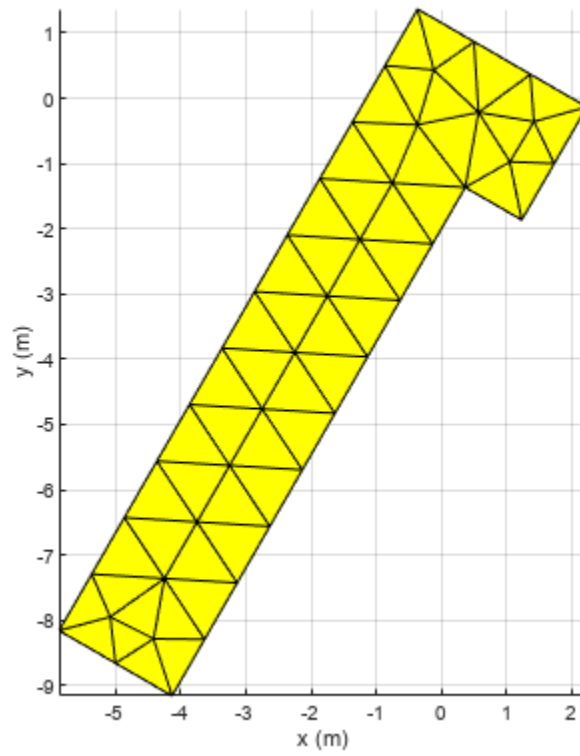
```
bend =  
  bendRightAngle with properties:  
      Name: 'myRightAnglebend'  
  ReferencePoint: [0 0]  
      Length: [10 2]  
      Width: [2 2]
```

```
show(bend)
```



Mesh the right-angle bend shape at a maximum edge length of 1 m.

```
mesh(bend,MaxEdgeLength=1)
```



Version History

Introduced in R2021b

See Also

bendMitered | bendCurved

curve

Create curved shape on X-Y plane

Description

Use the curve object to create a curved shape centered on the X-Y plane.

Creation

Syntax

```
curvedshape = curve  
curvedshape = curve(Name=Value)
```

Description

curvedshape = curve creates a curved shape centered on the X-Y plane.

curvedshape = curve(Name=Value) sets “Properties” on page 1-72 using one or more name-value arguments. For example, curve(ReferencePoint=[1 1]) creates a curved shape at the reference point [1 1]. Properties not specified retain their default values.

Properties

Name — Name of curved shape

'myCurve' (default) | character vector | string scalar

Name of the curved shape, specified as a character vector or string scalar.

Example: curvedshape = curve(Name='curvedshape1')

Data Types: char | string

ReferencePoint — Reference point

[0 0] (default) | two-element vector

Reference point of the curved shape in Cartesian coordinates, specified as a two-element vector.

Example: curvedshape = curve(ReferencePoint=[1 1])

Data Types: double

Radius — Radius of curved shape

0.0100 (default) | positive scalar

Radius of the curved shape, specified as a positive scalar in meters.

Example: curvedshape = curve(Radius=0.0300)

Data Types: double

Width — Width of curved shape

0.0020 (default) | positive scalar

Width of the curved shape, specified as a positive scalar in meters.

Example: `curvedshape = curve(Width=0.0030)`

Data Types: double

ArcAngle — Start angle and stop angle

[0 180] (default) | two-element vector

Start angle and stop angle in degrees, specified as a two-element vector.

Example: `curvedshape = curve(ArcAngle=[90 140])`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples**Create Default Curved Shape**

Create a curved shape with default properties.

`curved = curve``curved =``curve with properties:`

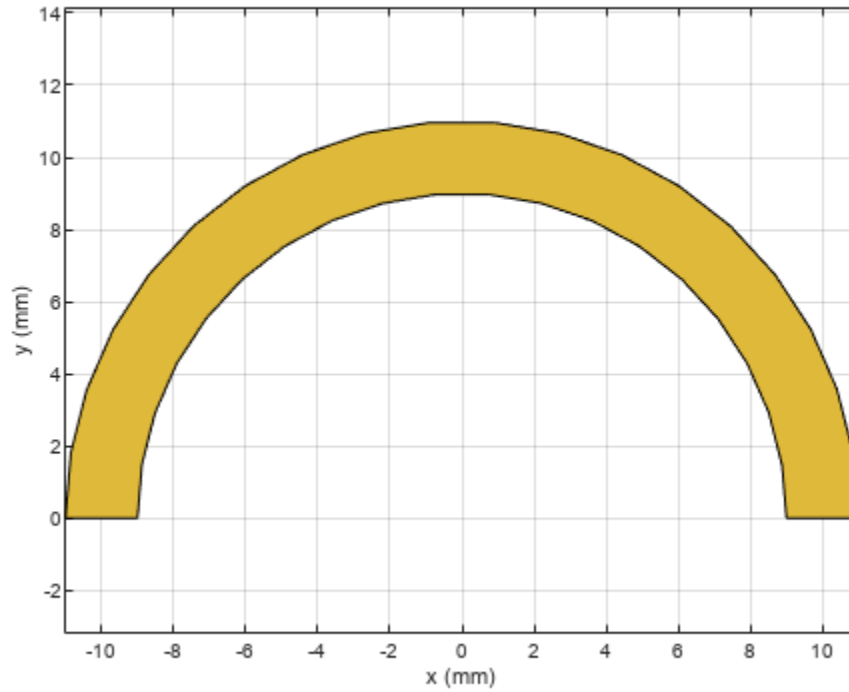
```

        Name: 'myCurve'
ReferencePoint: [0 0]
        Radius: 0.0100
        Width: 0.0020
        ArcAngle: [0 180]

```

View the curved shape.

```
show(curved)
```



Mesh Rotated Curved Shape

Create a curved shape with the of 8 m and width of 2 m.

```
curved = curve(Radius=8,Width=2);
```

Rotate the shape by 60 degrees.

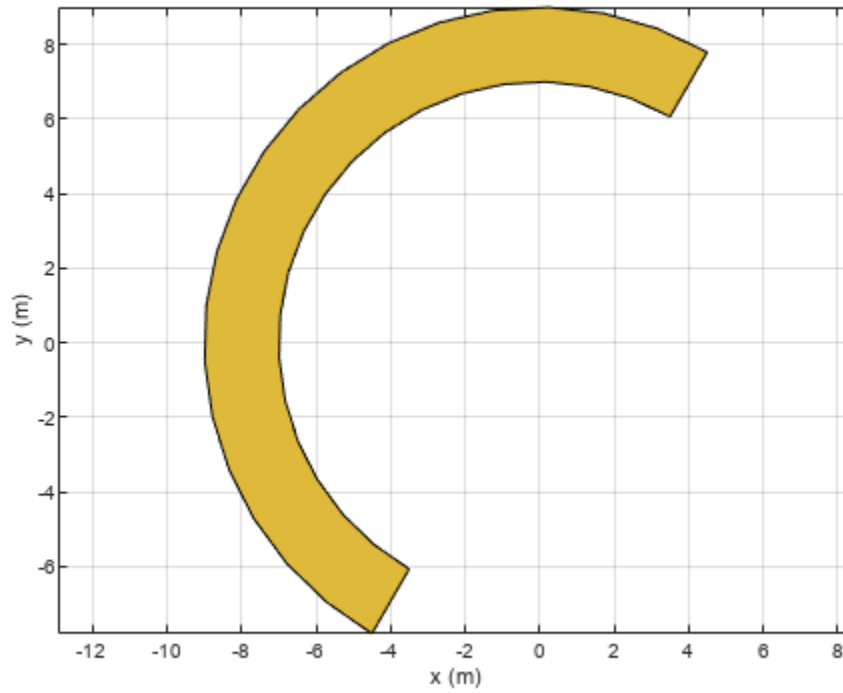
```
curved = rotateZ(curved,60)
```

```
curved =
```

```
curve with properties:
```

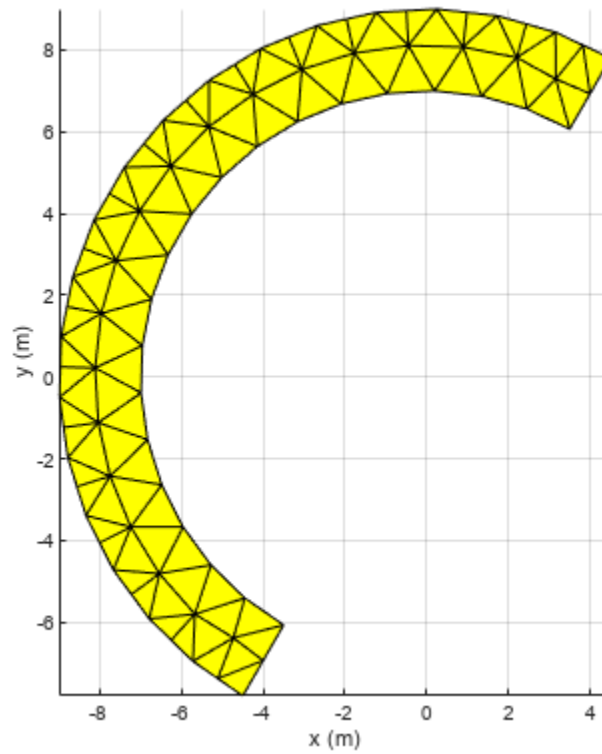
```
    Name: 'myCurve'  
ReferencePoint: [0 0]  
    Radius: 8  
    Width: 2  
ArcAngle: [0 180]
```

```
show(curved)
```



Mesh the curved shape at a maximum edge length of 1 m.

```
mesh(curved,MaxEdgeLength=1)
```



Version History

Introduced in R2021b

See Also

`ringAnnular` | `ringSquare` | `radial` | `delta`

ringAnnular

Create annular ring on X-Y plane

Description

Use the `ringAnnular` object to create an annular ring on the X-Y plane.

Creation

Syntax

```
ring = ringAnnular
ring = ringAnnular(Name=Value)
```

Description

`ring = ringAnnular` creates an annular ring on the X-Y plane.

`ring = ringAnnular(Name=Value)` sets “Properties” on page 1-77 using one or more name-value arguments. For example, `ringAnnular(Center=[1 1])` creates an annular ring shape centered at [1 1]. Properties not specified retain their default values.

Properties

Name — Name of annular ring

'myringAnnular' (default) | character vector | string scalar

Name of the annular ring, specified as a character vector or string scalar.

Example: `ring = ringAnnular(Name='ringannular1')`

Data Types: `char` | `string`

Center — Center of annular ring

[0 0] (default) | two-element vector

Center of the annular ring in Cartesian coordinates, specified as a two-element vector.

Example: `ring = ringAnnular(Center=[1 1])`

Data Types: `double`

InnerRadius — Inner radius of annular ring

0.0050 (default) | positive scalar

Inner radius of the annular ring, specified as a positive scalar in meters.

Example: `ring = ringAnnular(InnerRadius=0.006)`

Data Types: `double`

Width — Width of annular ring

0.0020 (default) | positive scalar

Width of the annular ring, specified as a positive scalar in meters.

Example: `ring = ringAnnular(Width=3)`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

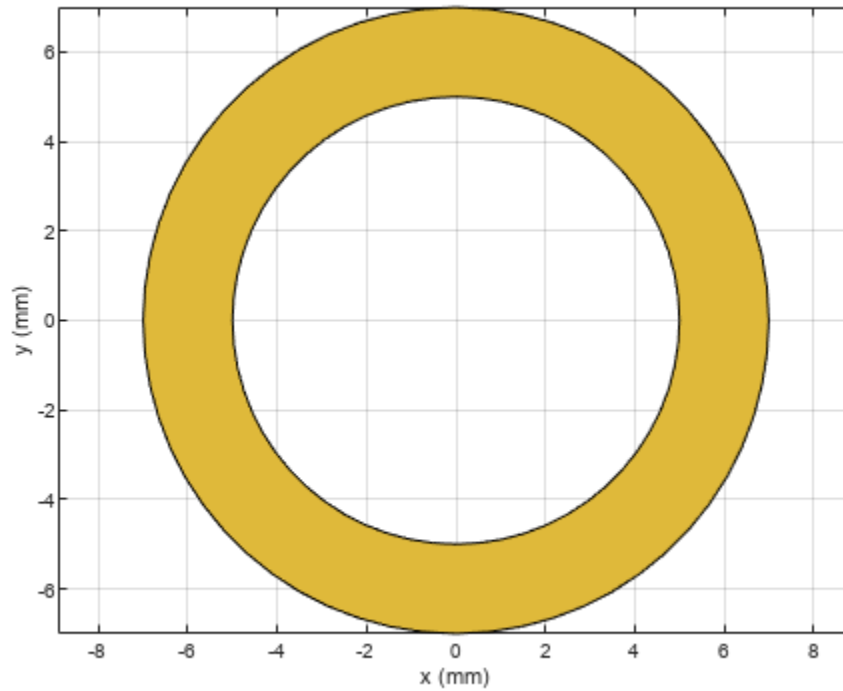
Examples**Create Default Annular Ring Shape**

Create an annular ring shape with default properties.

```
ring = ringAnnular
ring =
    ringAnnular with properties:
        Name: 'myringAnnular'
        Center: [0 0]
        Width: 0.0020
        InnerRadius: 0.0050
```

View the shape.

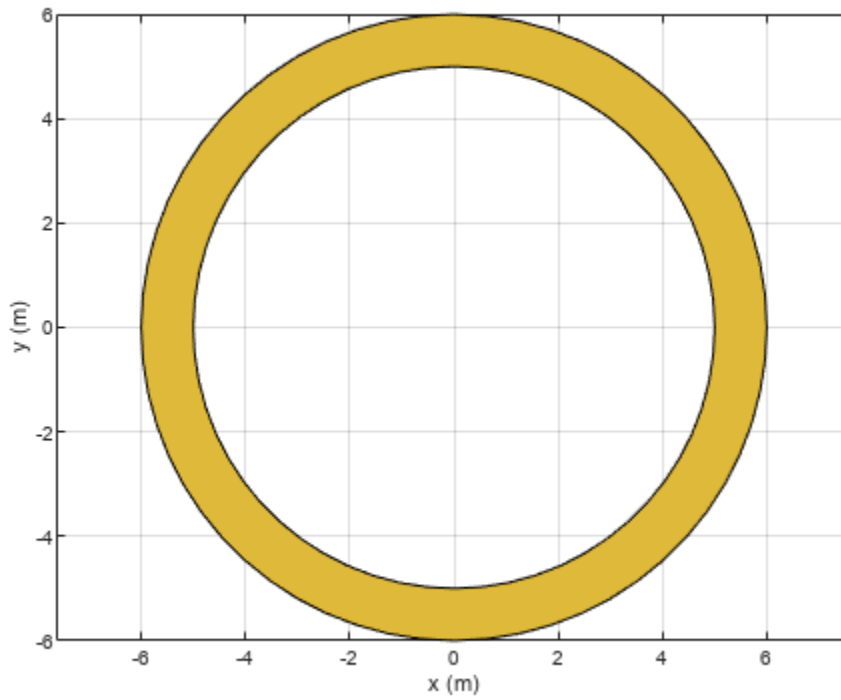
```
show(ring)
```



Mesh Annular Ring Shape

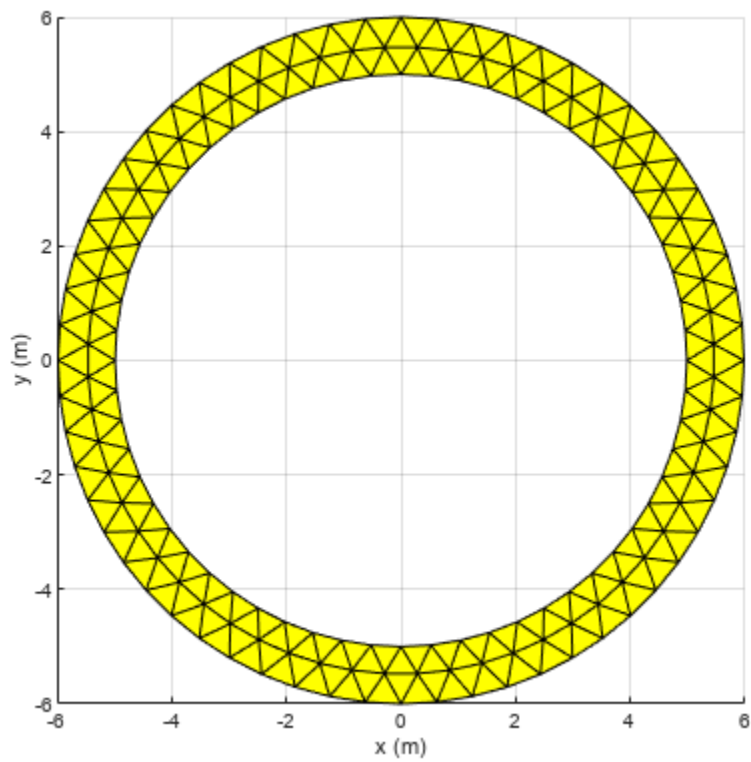
Create an annular ring shape of inner radius 5 m and width 1 m.

```
ring = ringAnnular(InnerRadius=5,Width=1);  
show(ring)
```



Mesh the ring at a maximum edge length of 1 m.

```
mesh(ring,MaxEdgeLength=1)
```



Version History

Introduced in R2021b

See Also

ringAnnular | ringSquare

ringSquare

Create square ring on X-Y plane

Description

Use the ringSquare object to create a square ring on the X-Y plane.

Creation

Syntax

```
ring = ringSquare  
ring = ringSquare(Name=Value)
```

Description

`ring = ringSquare` creates a square ring on the X-Y plane.

`ring = ringSquare(Name=Value)` sets “Properties” on page 1-82 using one or more name-value arguments. For example, `ringSquare(Center=[1 1])` creates an square ring shape centered at [1 1]. Properties not specified retain their default values.

Properties

Name — Name of square ring

'myringSquare' (default) | character vector | string scalar

Name of square ring, specified as a character vector or string scalar.

Example: `ring = ringSquare(Name='ringsquare1')`

Data Types: char | string

Center — Center of square ring

[0 0] (default) | two-element vector

Center of the square ring in Cartesian coordinates, specified as a two-element vector.

Example: `ring = ringAnnular(Center=[1 1])`

Data Types: double

InnerSide — Length of inner side

0.0050 (default) | positive scalar

Length of the inner side, specified as a positive scalar in meters.

Example: `ring = ringSquare(InnerSide=0.0060)`

Data Types: double

Width — Width of square ring

0.0020 (default) | positive scalar

Width of the square ring, specified as a positive scalar in meters.

Example: `ring = ringSquare(Width=0.0030)`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples**Create Default Square Ring Shape**

Create a square ring with default properties.

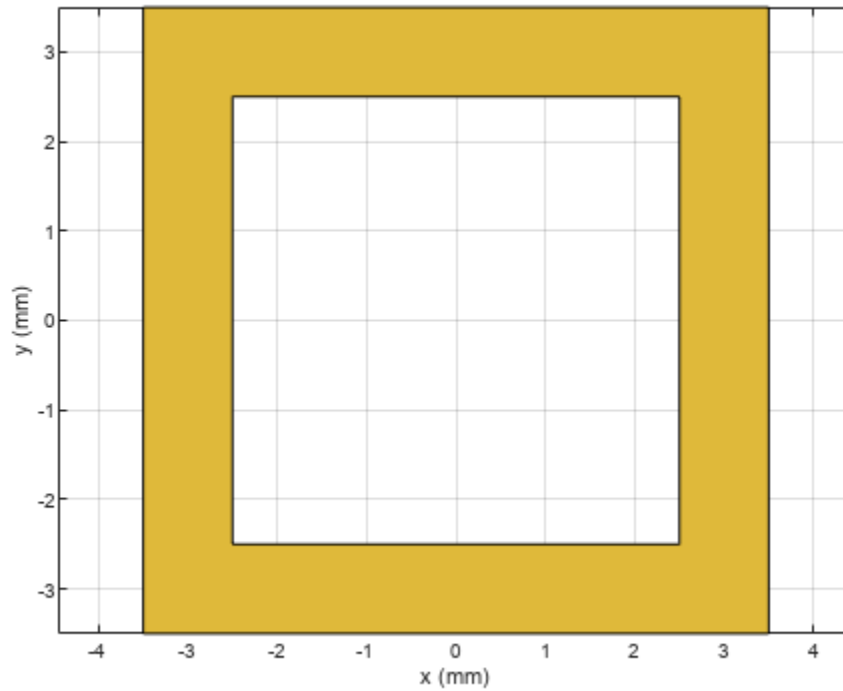
```
ring = ringSquare

ring =
    ringSquare with properties:

        Name: 'myringSquare'
        Center: [0 0]
        Width: 0.0020
        InnerSide: 0.0050
```

View the shape.

```
show(ring)
```



Mesh Square Ring Shape

Create a square ring with with the inner side length of 5 m and a width of 2 m.

```
ring = ringSquare(InnerSide=5,Width=2)
```

```
ring =
```

```
ringSquare with properties:
```

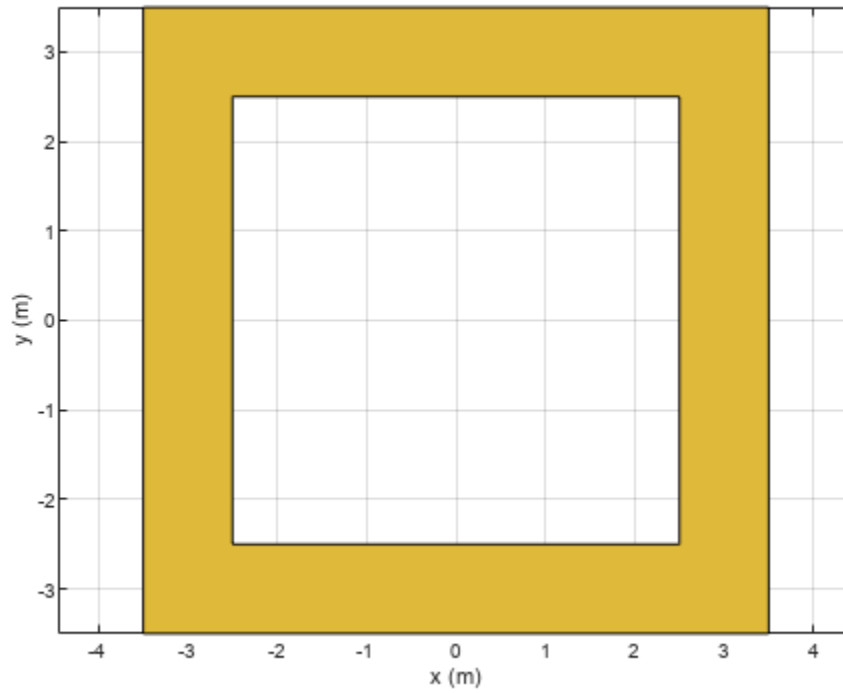
```
    Name: 'myringSquare'
```

```
    Center: [0 0]
```

```
    Width: 2
```

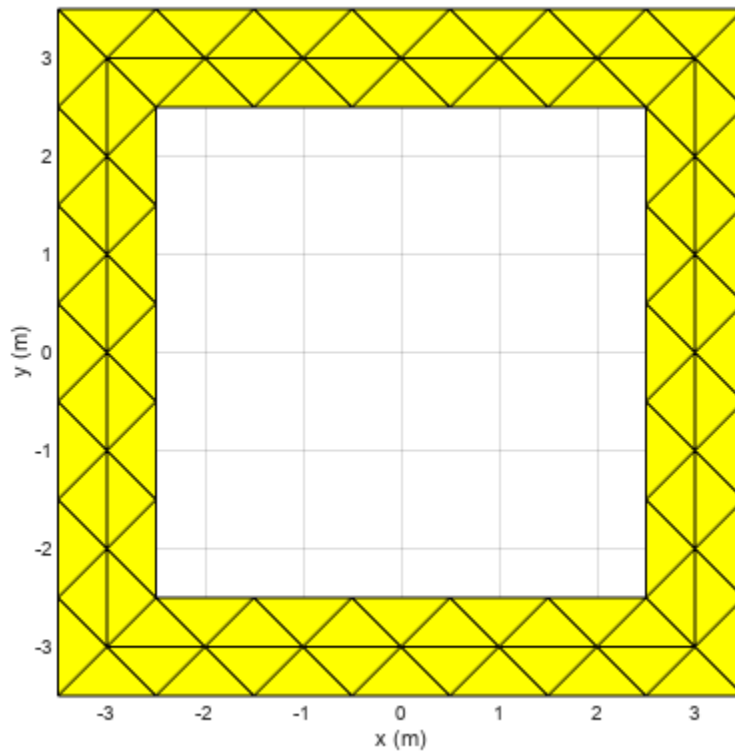
```
    InnerSide: 5
```

```
show(ring)
```

Mesh the square ring at a maximum edge length of 1 m.

```
mesh(ring,MaxEdgeLength=1)
```



Version History

Introduced in R2021b

See Also

`ringAnnular`

ubendCurved

Create U-bend with curved edges on X-Y plane

Description

Use the `ubendCurved` object to create a U-bend with curved edges on the X-Y plane.

Creation

Syntax

```
bend = ubendCurved
bend = ubendCurved(Name=Value)
```

Description

`bend = ubendCurved` creates a U-bend with curved edges on the X-Y plane.

`bend = ubendCurved(Name=Value)` sets “Properties” on page 1-87 using one or more name-value arguments. For example, `ubendCurved(ReferencePoint=[1 1])` creates a U-bend with curved edges at the reference point [1 1]. Properties not specified retain their default values.

Properties

Name — Name of curved U-bend

'myCurvedubend' (default) | character vector | string scalar

Name of the curved U-bend, specified as a character vector or a string scalar.

Example: `bend = ubendCurved(Name="ubendcurve1")`

Data Types: char | string

ReferencePoint — Reference point of curved U-bend

[0 0] (default) | two-element vector

Reference point of the curved U-bend in Cartesian coordinates, specified as a two-element vector.

Example: `bend = ubendCurved(ReferencePoint=[1 2])`

Data Types: double

Length — Length of curved U-bend

[0.0150 0.0050 0.0150] (default) | three-element vector

Length of the curved U-bend in meters, specified as a three-element vector of positive elements.

Example: `bend = ubendCurved(Length=[0.0050 0.0020 0.0050])`

Data Types: double

Width — Width of curved U-bend

[0.0050 0.0050 0.0050] (default) | three-element vector

Width of the curved U-bend in meters, specified as a three-element vector of positive elements.

Example: `bend = ubendCurved(Width=[0.0040 0.0040 0.0040])`

Data Types: double

CurveRadius — Radius of corner

0.0035 (default) | positive scalar

Radius of the corner in meters, specified as a positive scalar.

Example: `bend = ubendCurved(CurveRadius=0.0025)`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples**Create Default Curved U-Bend**

Create a curved U-bend with default properties.

```
curvedubend = ubendCurved
```

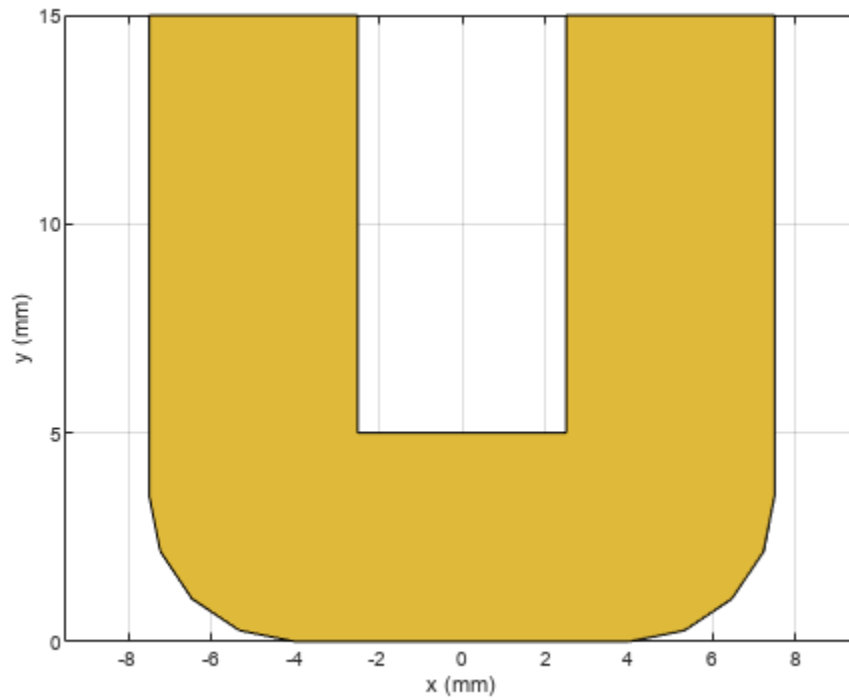
```
curvedubend =
```

```
    ubendCurved with properties:
```

```
        Name: 'myCurvedubend'  
ReferencePoint: [0 0]  
        Length: [0.0150 0.0050 0.0150]  
        Width: [0.0050 0.0050 0.0050]  
        CurveRadius: 0.0035
```

View the shape.

```
show(curvedubend)
```



Version History

Introduced in R2021b

See Also

[ubendMitered](#) | [ubendRightAngle](#)

ubendMitered

Create U-bend with mitered edges on X-Y plane

Description

Use the `ubendMitered` object to create a U-bend with mitered edges on the X-Y plane.

Creation

Syntax

```
bend = ubendMitered  
bend = ubendMitered(Name=Value)
```

Description

`bend = ubendMitered` creates a U-bend with mitered edges on the X-Y plane.

`bend = ubendMitered(Name=Value)` sets “Properties” on page 1-90 using one or more name-value arguments. For example, `ubendMitered(ReferencePoint=[1 1])` creates a mitered U-bend at the reference point `[1 1]`. Properties not specified retain their default values.

Properties

Name — Name of mitered U-bend

'myMiteredubend' (default) | character vector | string scalar

Name of the mitered U-bend, specified as a character vector or a string scalar.

Example: `bend = ubendMitered(Name="ubendmitered1")`

Data Types: `char` | `string`

ReferencePoint — Reference point of mitered U-bend

[0 0] (default) | two-element vector

Reference point of the mitered U-bend in Cartesian coordinates, specified as a two-element vector.

Example: `bend = ubendMitered(ReferencePoint=[1 2])`

Data Types: `double`

Length — Length of mitered U-bend

[0.0150 0.0050 0.0150] (default) | three-element vector

Length of the mitered U-bend in meters, specified as a three-element vector of positive elements.

Example: `bend = ubendMitered(Length=[0.0250 0.0030 0.0250])`

Data Types: `double`

Width — Width of mitered U-bend

[0.0050 0.0050 0.0050] (default) | three-element vector

Width of the mitered U-bend in meters, specified as a three-element vector of positive elements.

Example: `bend = ubendMitered(Width=[0.0060 0.0060 0.0060])`

Data Types: double

MiterDiagonal — Length of miter diagonal

0.0035 (default) | positive scalar

Length of the miter diagonal in meters, specified as a positive scalar.

Example: `bend = ubendMitered(MiterDiagonal=0.0060)`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples**Create Default Mitered U-Bend**

Create a mitered U-bend with default properties.

```
bend = ubendMitered
```

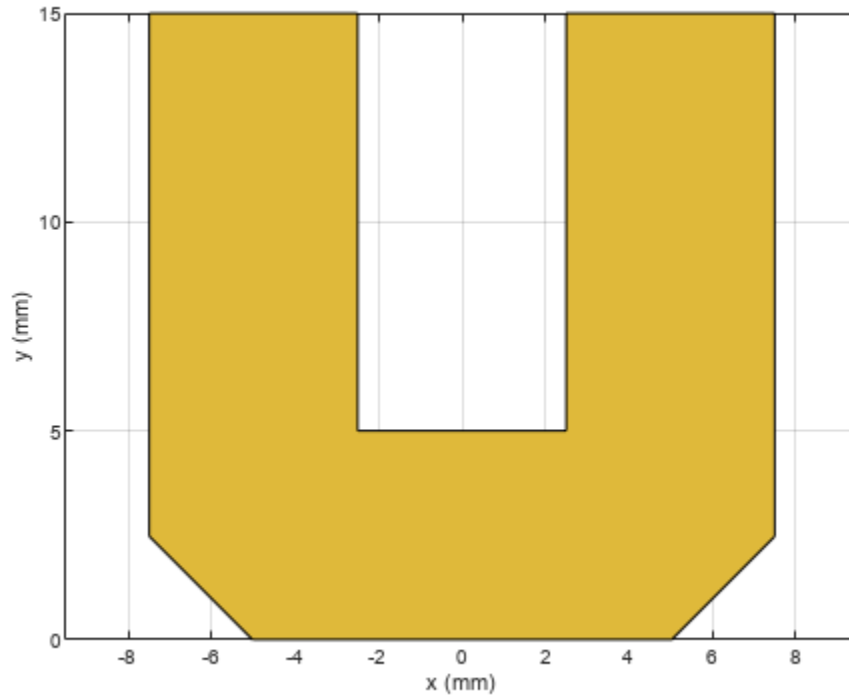
```
bend =
```

```
    ubendMitered with properties:
```

```
        Name: 'myMiteredubend'
ReferencePoint: [0 0]
        Length: [0.0150 0.0050 0.0150]
        Width: [0.0050 0.0050 0.0050]
MiterDiagonal: 0.0035
```

View the shape.

show(bend)



Version History

Introduced in R2021b

See Also

[ubendCurved](#) | [ubendRightAngle](#)

ubendRightAngle

Create right-angle U-bend shape on X-Y plane

Description

Use the `ubendRightAngle` object to create a right-angle U-bend shape on the X-Y plane.

Creation

Syntax

```
bend = ubendRightAngle
bend = ubendRightAngle(Name=Value)
```

Description

`bend = ubendRightAngle` creates a right-angle U-bend shape on the X-Y plane.

`bend = ubendRightAngle(Name=Value)` sets “Properties” on page 1-93 using one or more name-value arguments. For example, `ubendRightAngle(ReferencePoint=[1 1])` creates a right-angle U-bend at the reference point [1 1]. Properties not specified retain their default values.

Properties

Name — Name of right-angle U-bend

'myRightAngleubend' (default) | character vector | string scalar

Name of the right-angle U-bend, specified as a character vector or a string scalar.

Example: `bend = ubendRightAngle(Name="ubendrightangle1")`

Data Types: `char` | `string`

ReferencePoint — Reference point of right-angle U-bend

[0 0] (default) | two-element vector

Reference point of the right-angle U-bend in Cartesian coordinates, specified as a two-element vector.

Example: `bend = ubendRightAngle(ReferencePoint=[1 2])`

Data Types: `double`

Length — Length of right-angle U-bend

[0.0150 0.0050 0.0150] (default) | three-element vector

Length of the right-angled U-bend in meters, specified as a three-element vector of positive elements.

Example: `bend = ubendRightAngle(Length=[0.0250 0.0150 0.0250])`

Data Types: `double`

Width — Width of right-angle U-bend

[0.0050 0.0050 0.0050] (default) | three-element vector

Width of the right-angle U-bend in meters, specified as a three-element vector of positive values.

Example: `bend = ubendRightAngle(Width=[0.0150 0.0150 0.0010])`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples**Create Default Right-Angle U-Bend**

Create a right-angle U-bend with default properties.

```
bend = ubendRightAngle
```

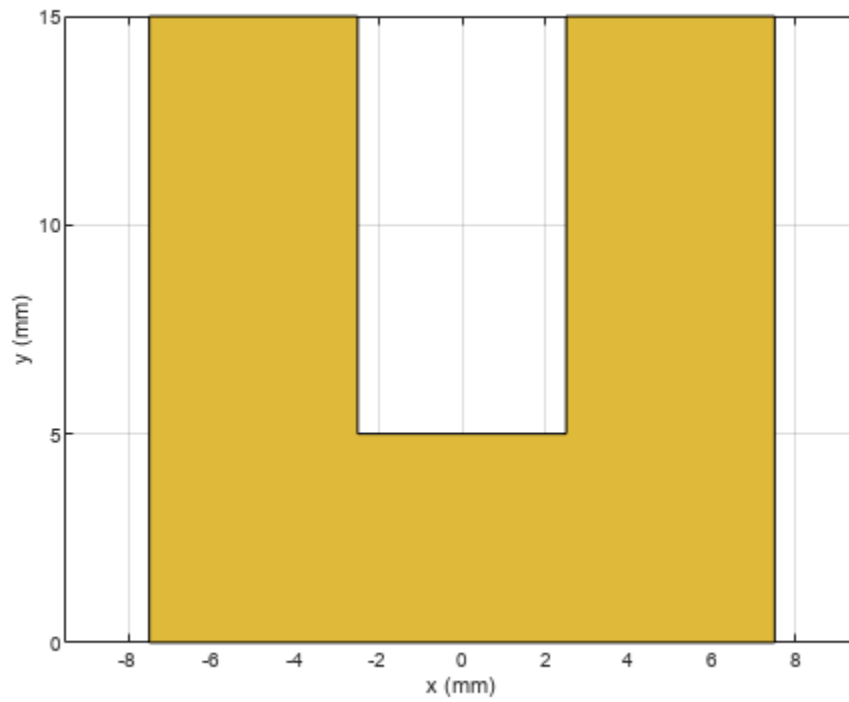
```
bend =
```

```
    ubendRightAngle with properties:
```

```
        Name: 'myRightAngleubend'  
    ReferencePoint: [0 0]  
        Length: [0.0150 0.0050 0.0150]  
        Width: [0.0050 0.0050 0.0050]
```

View the shape.

```
show(bend)
```



Version History

Introduced in R2021b

See Also

[ubendCurved](#) | [ubendMitered](#)

tracePoint

Create custom line trace based on specified X and Y coordinates

Description

Use the `tracePoint` object to create a custom line trace by tracing a line along the specified X and Y coordinates.

Creation

Syntax

```
trace = tracePoint  
trace = tracePoint(Name=Value)
```

Description

`trace = tracePoint` creates a line trace using default properties.

`trace = tracePoint(Name=Value)` sets “Properties” on page 1-96 using one or more name-value arguments. For example, `tracePoint(Width=0.0050)` creates a line trace with the width of 0.0050. Properties not specified retain their default values.

Properties

Name — Name of custom line trace

'tracePoint' (default) | character vector | string scalar

Name of the custom line trace, specified as a character vector or string scalar.

Example: `trace = tracePoint(Name='tracepoint1')`

Data Types: `char` | `string`

TracePoints — Coordinates of custom line trace

10-by-2-array (default) | *n*-by-2-array

Coordinates of custom line trace, specified as a *n*-by-2-array of X and Y coordinates.

Example: `trace = tracePoint(TracePoints=[0 0;0 -10;6 -10;6 0])`

Data Types: `double`

Width — Width of line trace

0.002 (default) | positive scalar

Width of the line trace, specified as a positive scalar in meters. This value is applied to all line segments in the custom trace.

Example: `trace = tracePoint(Width=0.005)`

Data Types: double

Corner — Corner where two line segments interface

"Sharp" (default) | "Miter" | "Smooth"

Corner where two line segments interface, specified as either "Sharp", "Miter", or "Smooth". To apply the same value to all corners, specify a string scalar. For a different value for all corners, specify a (n-2)-by-1 vector of strings.

Example: `trace = tracePoint(Corner="Miter")`

Data Types: string

Object Functions

add	Boolean unite operation on two RF PCB shapes
subtract	Boolean subtraction operation on two RF PCB shapes
intersect	Boolean intersection operation on two RF PCB shapes
plus	Shape1 + Shape2 for RF PCB shapes
minus	Shape1 - Shape2 for RF PCB shapes
and	Shape1 & Shape2 for RF PCB shapes
area	Calculate area of RF PCB shape in square meters
rotate	Rotate RF PCB shape about defined axis
rotateX	Rotate RF PCB shape about x-axis
rotateY	Rotate RF PCB shape about y-axis and angle
rotateZ	Rotate RF PCB shape about z-axis
translate	Move RF PCB shape to new location
scale	Change size of RF PCB shape by fixed amount

Examples

Create Default Custom Line

Create a custom line using default properties.

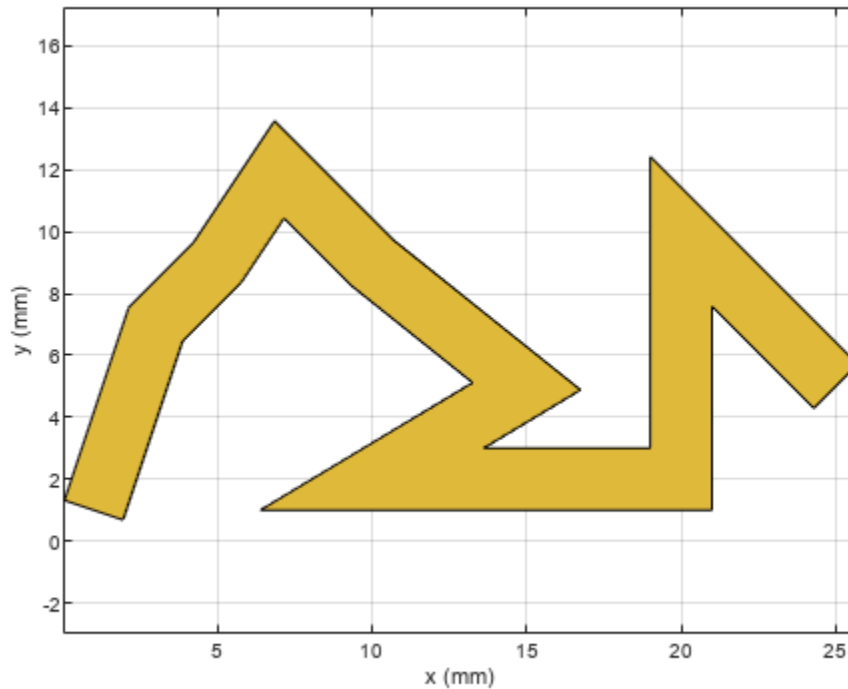
```
customLine = tracePoint

customLine =
    tracePoint with properties:

        Name: 'mytracePoint'
        TracePoints: [10x2 double]
        Width: 0.0020
        Corner: "Sharp"
```

View the trace.

```
show(customLine)
```



Rotate and Mesh Custom Line

Create a custom line trace using default properties.

```
customLine = tracePoint;
```

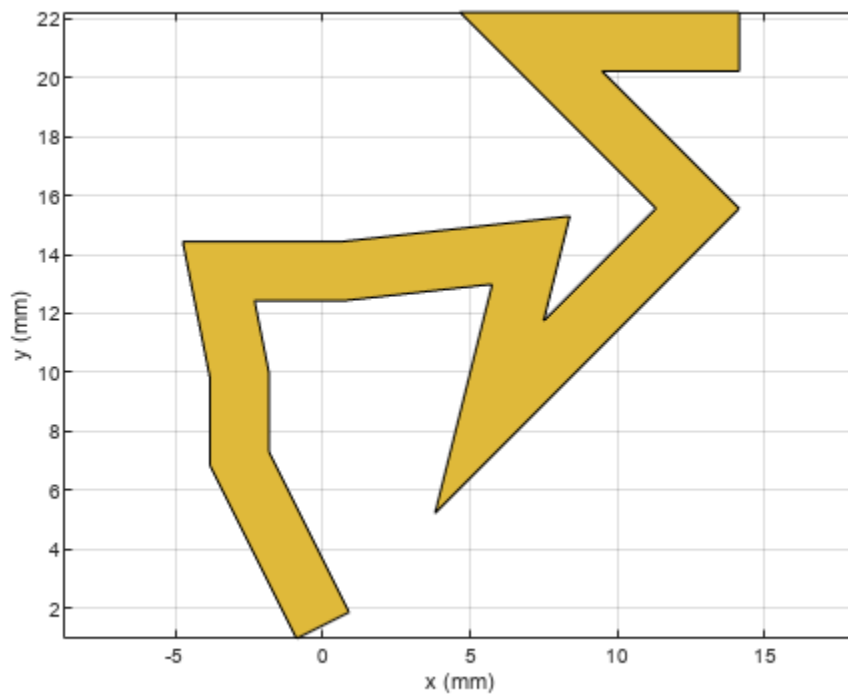
Rotate the trace by 45 degrees along the Z-axis.

```
customLine = rotateZ(customLine,45)
```

```
customLine =  
  tracePoint with properties:  
  
    Name: 'mytracePoint'  
  TracePoints: [10x2 double]  
    Width: 0.0020  
   Corner: "Sharp"
```

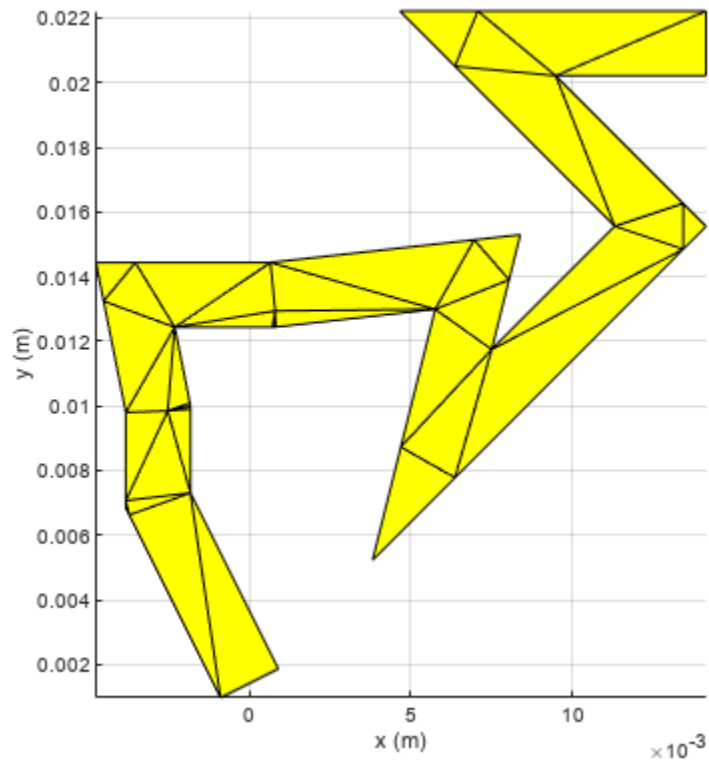
View the trace.

```
show(customLine)
```



Mesh the custom line trace at a maximum edge length of 1 m.

```
mesh(customLine,MaxEdgeLength=1)
```



Create Custom Trace with Smooth Corners

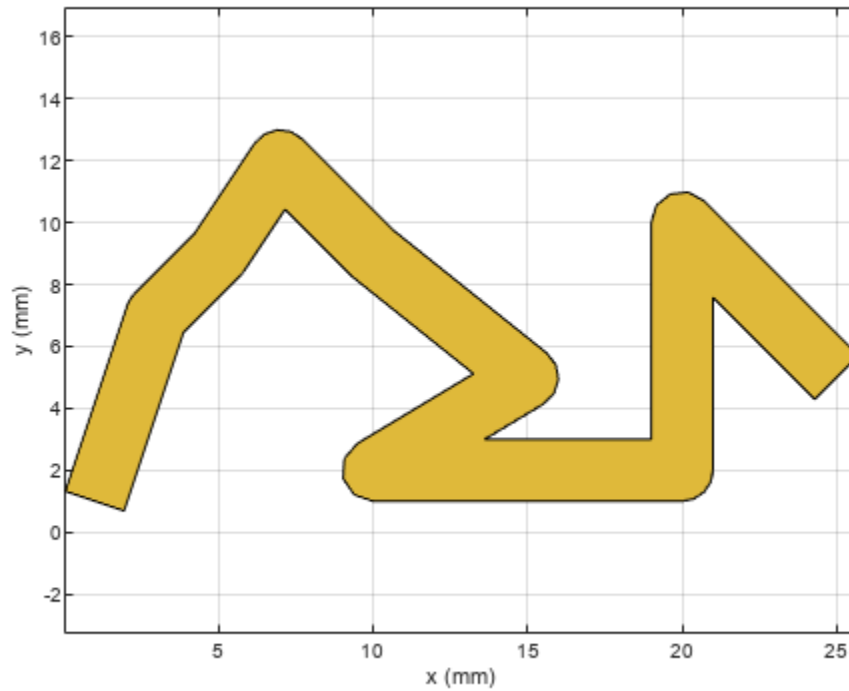
Create a custom trace line with smooth corners.

```
customLine = tracePoint(Name='tracepoint',Corner="Smooth")
```

```
customLine =  
  tracePoint with properties:
```

```
    Name: 'tracepoint'  
  TracePoints: [10x2 double]  
    Width: 0.0020  
    Corner: "Smooth"
```

```
show(customLine)
```

Version History

Introduced in R2021b

See Also

[traceLine](#) | [traceCross](#) | [traceTee](#) | [traceRectangular](#) | [traceSpiral](#)

traceLine

Create line trace

Description

Use the `traceLine` object to create a line trace. You can use this object to create lines of different lengths and different angles

Creation

Syntax

```
trace = traceLine
trace = traceLine(Name=Value)
```

Description

`trace = traceLine` creates a line trace using default properties.

`trace = traceLine(Name=Value)` sets properties using one or more name-value arguments. For example, `traceLine('StartPoint', [1 1])` creates a line trace shape with the starting point of [1 1]. Properties not specified retain their default values.

Properties

Name — Name of line trace

'traceLine' (default) | character vector | string scalar

Name of the line trace, specified as a character vector or string scalar.

Example: `customtrace = traceLine(Name=traceline1)`

Data Types: `char` | `string`

StartPoint — Start point of line trace

[0 0] (default) | two-element vector

Start point of the line trace in Cartesian coordinates, specified as a two-element vector.

Example: `customtrace = traceLine(StartPoint=[1 1])`

Data Types: `double`

Length — Length of line trace

[0.0200 0.0200 0.0200 0.0150] (default) | *n*-by-1 vector

Length of line trace, specified as an *n*-by-1 vector in meters. Each element represents the length of a line segment.

Example: `customtrace = traceLine(Length=[0.0100 0.0100 0.0100 0.0500])`

Data Types: double

Width — Width of line trace

0.0050 (default) | n -by-1 vector

Width of the line trace, specified as a scalar or an n -by-1 vector in meters. Each element represents the length of a line segment

Example: `customtrace = traceLine(Width=[0.0040 0.0040 0.0040 0.0050])`

Data Types: double

Angle — Angle of line trace

[90 0 -90 45] (default) | n -by-1 vector

Angle of the line trace, specified as an n -by-1 vector in degrees. Each element represents an angle of a line segment.

Example: `customtrace = traceLine(Angle=[40 10 -40 35])`

Data Types: double

Corner — Corner where two line segments interface

"Sharp" (default) | "Miter" | "Smooth"

Corner where two line segments interface, specified as either "Sharp", "Miter", or "Smooth". To apply the same value to all corners, specify a string scalar. For a different value for all corners, specify a $(n-2)$ -by-1 vector of strings.

Example: `trace = traceLine(Corner="Miter")`

Data Types: string

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples

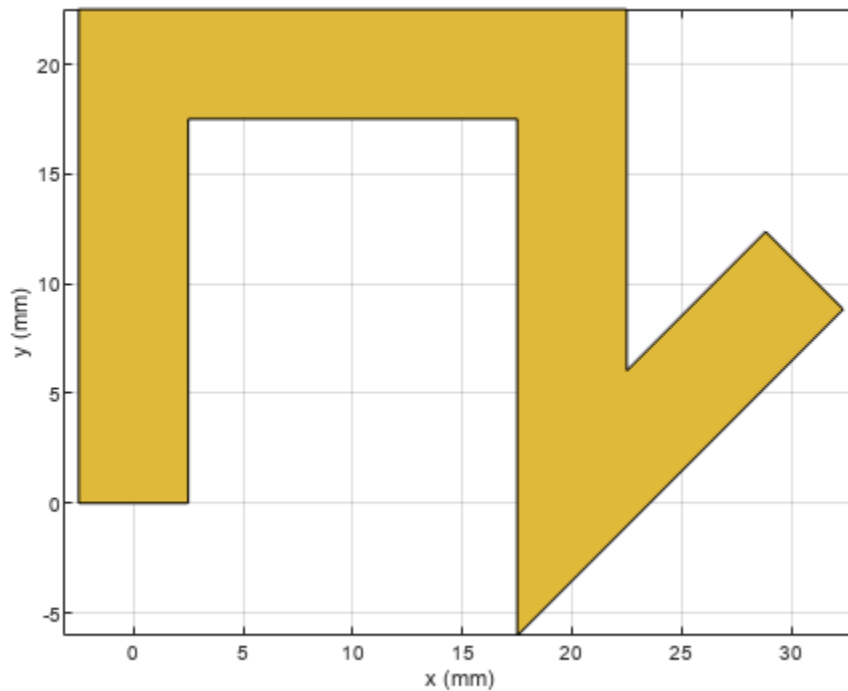
Create Default Custom Line Trace

Create a custom line trace with default properties.

```
customLine = traceLine
customLine =
  traceLine with properties:
    Name: 'mytraceLine'
    StartPoint: [0 0]
    Length: [0.0200 0.0200 0.0200 0.0150]
    Width: 0.0050
    Angle: [90 0 -90 45]
    Corner: "Sharp"
```

View the trace.

```
show(customLine)
```



Rotate and Mesh Line Trace

Create a line trace.

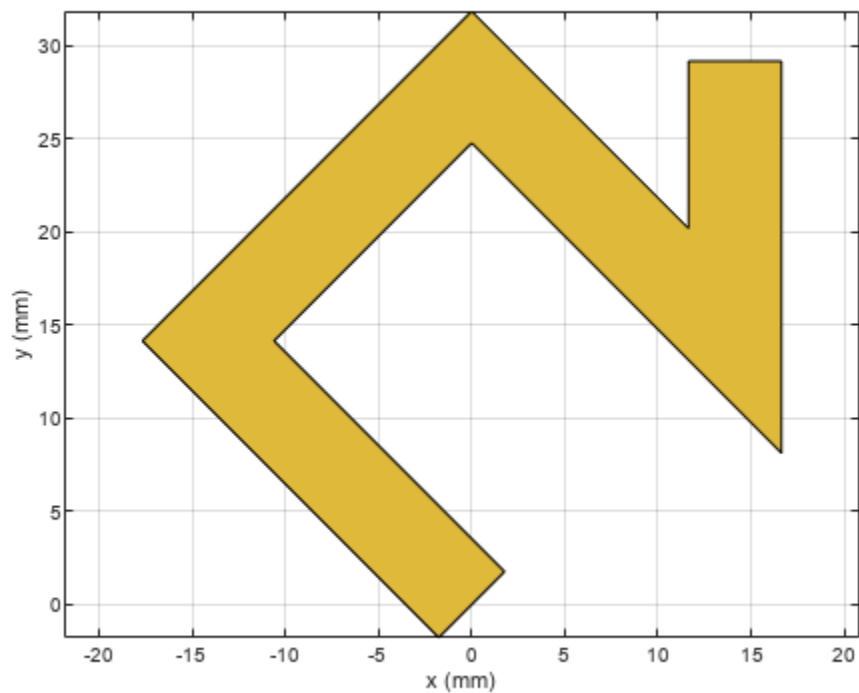
```
customLine = traceLine;
```

Rotate the trace by 45 degrees along the Z-axis.

```
customLine = rotateZ(customLine,45)
```

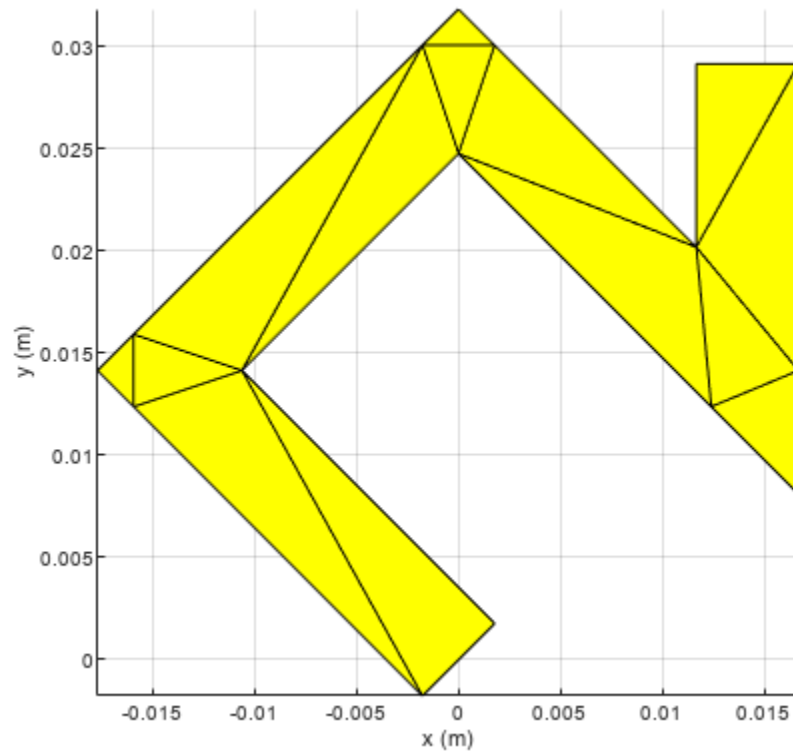
```
customLine =  
  traceLine with properties:  
  
    Name: 'mytraceLine'  
  StartPoint: [0 0]  
  Length: [0.0200 0.0200 0.0200 0.0150]  
  Width: 0.0050  
  Angle: [90 0 -90 45]  
  Corner: "Sharp"
```

```
show(customLine)
```



Mesh the line trace at a maximum edge length of 1 m.

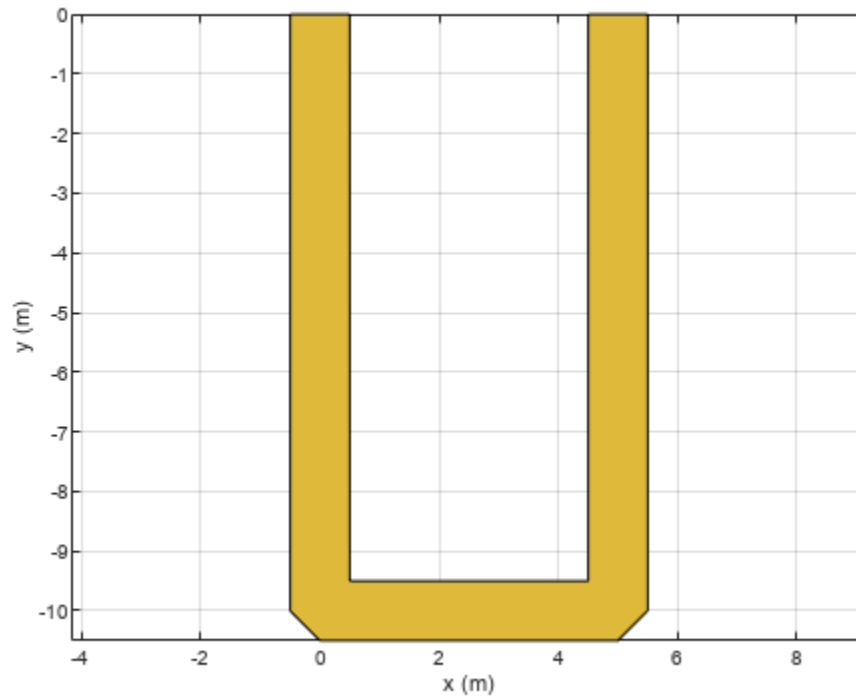
```
mesh(customLine,MaxEdgeLength=1)
```



U-Shaped Line Trace

Create and view a U-shaped line trace with mitered bends and a width of 1 m.

```
Ushapeline = traceLine;  
Ushapeline.Length = [10 5 10];  
Ushapeline.Angle = [-90 0 90];  
Ushapeline.Width = 1;  
Ushapeline.Corner = 2;  
show(Ushapeline);
```



Version History

Introduced in R2021b

See Also

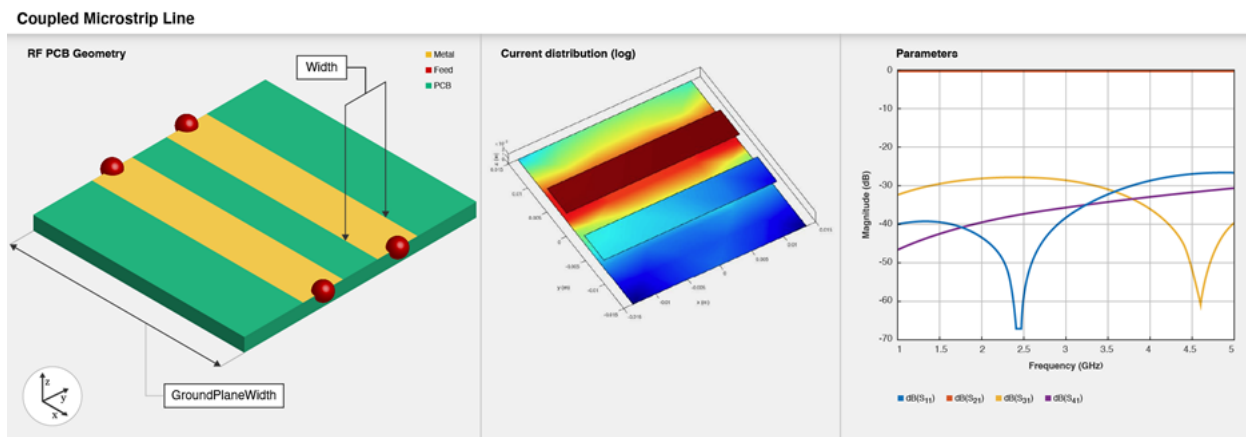
[traceCross](#) | [traceTee](#) | [traceRectangular](#) | [traceSpiral](#) | [tracePoint](#)

coupledMicrostripLine

Create coupled microstrip transmission line

Description

Use the `coupledMicrostripLine` object to create a coupled microstrip transmission line. Coupled microstrip transmission lines are used to design directional couplers and filters. The combination of even and odd mode impedances determines the coupling ratio between the direct arm and the coupled arm.



Creation

Syntax

```
coupledmicrostrip = coupledMicrostripLine
coupledmicrostrip = coupledMicrostripLine(Name=Value)
```

Description

`coupledmicrostrip = coupledMicrostripLine` creates a default coupled microstrip transmission line with a Teflon substrate. The default properties are for a resonating frequency of 1.5 GHz.

`coupledmicrostrip = coupledMicrostripLine(Name=Value)` sets “Properties” on page 1-108 using one or more name-value arguments. For example, `coupledMicrostripLine(Length=0.0300)` creates a coupled microstrip transmission line of length 0.0300 meters.

Properties

Length — Length of coupled microstrip line

0.0271 (default) | positive scalar

Length of the coupled microstrip line in meters, specified as a positive scalar.

Example: `coupledmicrostrip = coupledMicrostripLine(Length=0.0300)`

Data Types: double

Width — Width of coupled microstrip line

0.0051 (default) | positive scalar

Width of the coupled microstrip line in meters, specified as a positive scalar.

Example: `coupledmicrostrip = coupledMicrostripLine(Width=0.0041)`

Data Types: double

Spacing — Distance between the direct arm and the coupled arm

0.0046 (default) | positive scalar

Distance between the direct arm and the coupled arm of the coupled microstrip transmission line, specified as a positive scalar in meters.

Example: `coupledmicrostrip = coupledMicrostripLine(Spacing=0.00300)`

Data Types: double

Height — Height of coupled microstrip line

0.0016 (default) | positive scalar

Height of the coupled microstrip line from the ground plane, specified as a positive scalar in meters. In the case of a multilayer substrate, use the `Height` property to create a coupled microstrip line at the interface of the two dielectrics.

Example: `coupledmicrostrip = coupledMicrostripLine(Height=0.0023)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0300 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `coupledmicrostrip = coupledMicrostripLine(GroundPlaneWidth=0.0400)`

Data Types: double

Substrate — Type of dielectric material

'Teflon' (default) | dielectric object

Type of the dielectric material used as a substrate, specified as a dielectric object. The thickness of the default dielectric material Teflon is 0.0016 m or the same as the `Height` property.

Example: `d = dielectric("FR4"); coupledmicrostrip = coupledMicrostripLine(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'PEC' (default) | metal object

Type of metal used in the conducting layers, specified as a metal object.

```
Example: m = metal("PEC"); coupledmicrostrip
=coupledMicrostripLine(Conductor=m)
```

Data Types: string | char

Object Functions

charge	Calculate and plot charge distribution
current	Calculate and plot current distribution
design	Design coupled microstrip transmission line around particular frequency
feedCurrent	Calculate current at feed port
getZ0	Calculate characteristic impedance of transmission line
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples

Default Coupled Microstrip Line

Create a default coupled microstrip line.

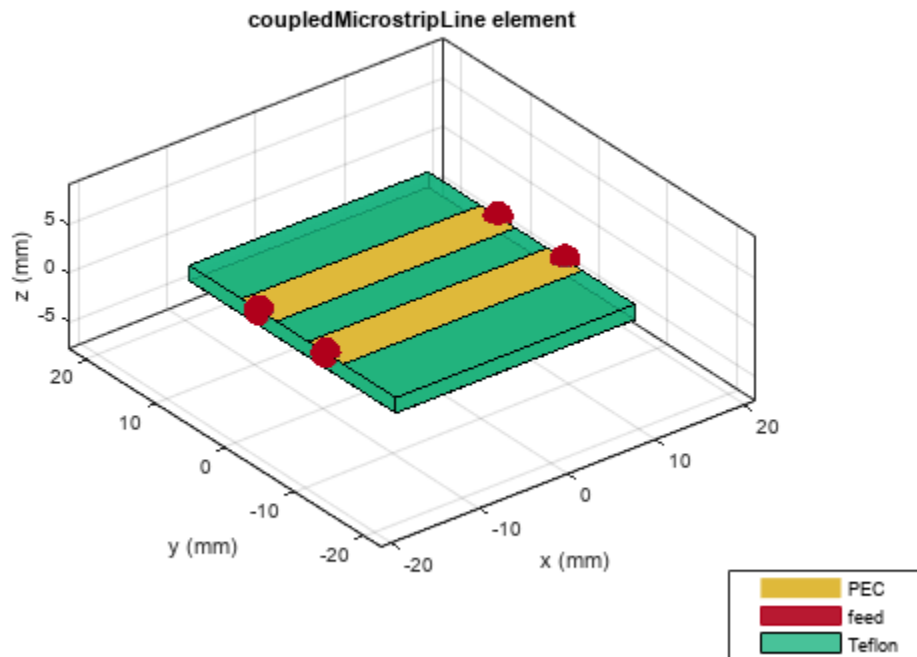
```
cml = coupledMicrostripLine
```

```
cml =  
  coupledMicrostripLine with properties:
```

```
      Length: 0.0271  
      Width: 0.0051  
      Spacing: 0.0046  
      Height: 0.0016  
GroundPlaneWidth: 0.0300  
      Substrate: [1x1 dielectric]  
      Conductor: [1x1 metal]
```

View the coupled microstrip line.

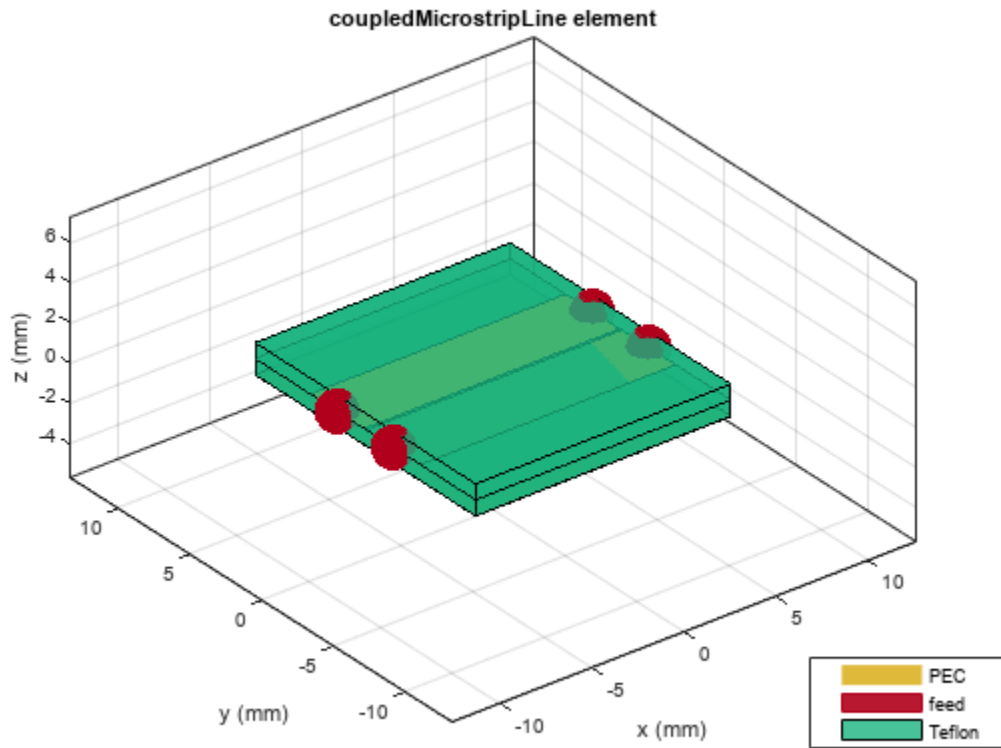
```
show(cml)
```



Multilayer Coupled Microstrip Line

Design and view a coupled microstrip line at the interface of a multilayer dielectric.

```
cml = design(coupledMicrostripLine,4e9,Z0e=75,Z0o=36);
cml.Substrate = dielectric(Name=["Teflon","Teflon"],EpsilonR=[2.1 2.1], ...
    LossTangent=[0 0],Thickness=[0.8e-3 0.8e-3]);
cml.Height = 0.8e-3;
show(cml)
```



Plot the current and charge distribution on the transmission line.

current(cml,4e9)

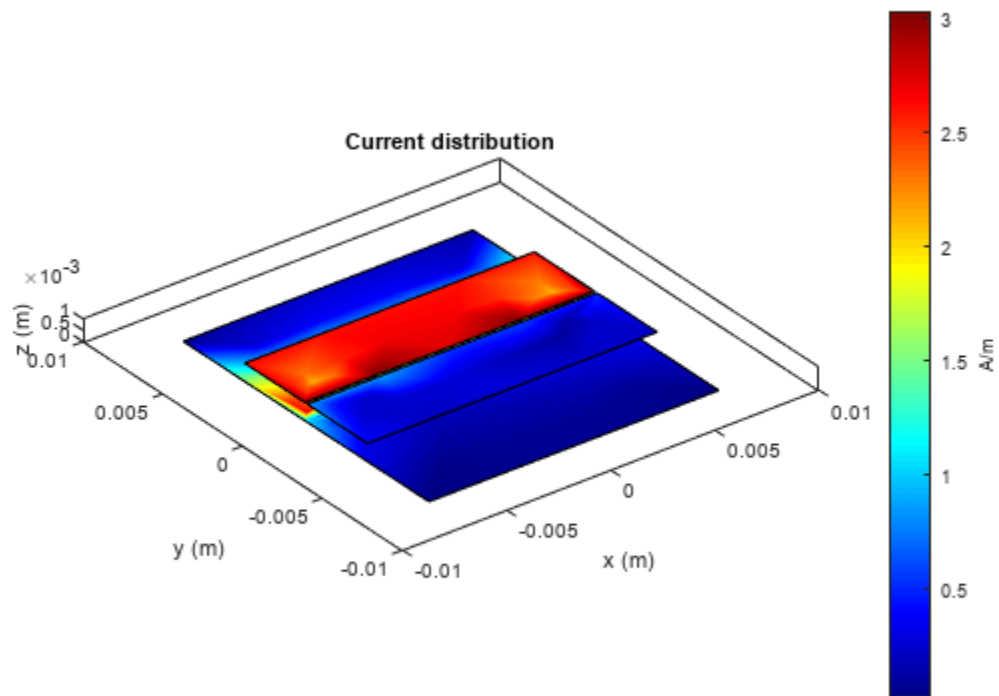
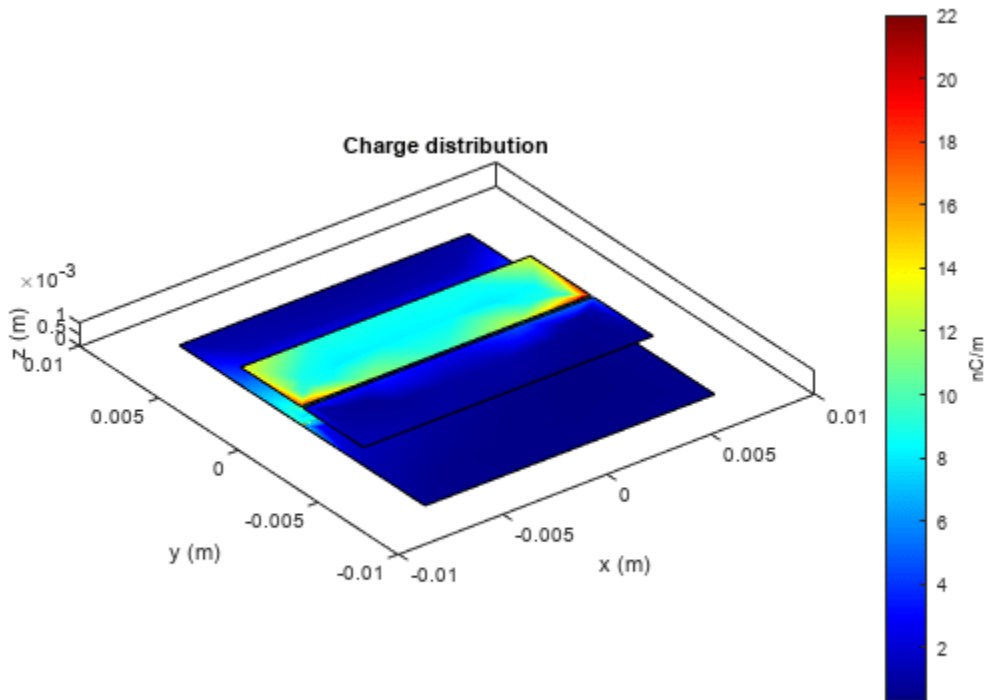


figure
charge(cml,4e9)



More About

Parametric Analysis

Use the design function to change the even impedance (Z_{oe}) and the odd impedance (Z_{oo}) of the coupled microstrip line.

- Increasing the difference between the even impedance (Z_{oe}) and the odd impedance (Z_{oo}) decreases the distance between the lines.
- Increasing the difference between Z_{oe} and Z_{oo} increases the power at the coupled ports.
- The impedance of the coupled lines Z_o is the geometric mean of Z_{oe} and Z_{oo} . If the even and odd impedance values do not satisfy this condition, then calculate the S-parameters using Z_o to get a proper match.

Version History

Introduced in R2021b

References

- [1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

[2] "Microwaves101 | Coupled Line Couplers." Accessed July 7, 2021. <https://www.microwaves101.com/encyclopedias/coupled-line-couplers>.

See Also

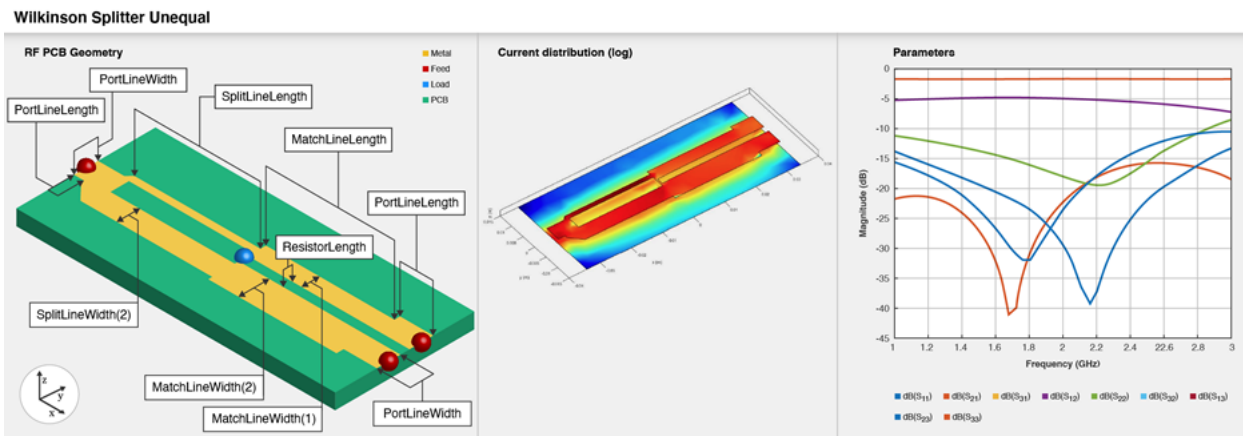
coplanarWaveguide | microstripLine

wilkinsonSplitterUnequal

Create unequal Wilkinson splitter

Description

Use the `wilkinsonSplitterUnequal` object to create an unequal Wilkinson power splitter. You can use the unequal Wilkinson splitter to divide power unequally between two output ports. Unequal splitters are also used to feed power to antenna arrays for beam shaping.



To analyze the behavioral model for the unequal Wilkinson power splitter, set the `Behavioral` property in the `sparameters` to `true` or `1`.

Creation

Syntax

```
splitter = wilkinsonSplitterUnequal
splitter = wilkinsonSplitterUnequal(Name=Value)
```

Description

`splitter = wilkinsonSplitterUnequal` creates an unequal Wilkinson splitter with a Teflon substrate. The default properties are for a resonating frequency of 1 GHz.

`splitter = wilkinsonSplitterUnequal(Name=Value)` sets “Properties” on page 1-117 using one or more name-value arguments. For example, `wilkinsonSplitterUnequal(PortLineLength=0.0300)` creates a Wilkinson splitter with an input and output line length of 0.0300 meters. Properties not specified retain their default values.

Properties

PortLineLength — Length of input and output line

0.0070 (default) | positive scalar

Length of the input and the output line in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitterUnequal(PortLineLength=0.0070)`

Data Types: double

PortLineWidth — Width of input and output line

0.0051 (default) | positive scalar

Width of the input and the output line in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitterUnequal(PortLineWidth=0.0070)`

Data Types: double

SplitLineLength — Length of 70-ohm line

0.0279 (default) | positive scalar

Length of the 70-ohm line in meters, specified as a positive scalar. The typical length of a Wilkinson splitter is $\lambda/4$.

Example: `splitter = wilkinsonSplitterUnequal(SplitLineLength=0.0570)`

Data Types: double

SplitLineWidth — Width of 70-ohm line

[0.0014 0.0049] (default) | two-element vector

Width of the 70-ohm line in meters, specified as a two-element vector of positive elements.

Example: `splitter = wilkinsonSplitterUnequal(SplitLineWidth=[0.00780 0.00890])`

Data Types: double

MatchLineLength — Length of output matching line

0.0277 (default) | positive scalar

Length of the output matching line in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitterUnequal(MatchLineLength=0.0780)`

Data Types: double

MatchLineWidth — Width of output matching line

[0.0039 0.0066] (default) | two-element vector

Width of the output matching line in meters, specified as a two-element vector of positive elements.

Example: `splitter = wilkinsonSplitterUnequal(MatchLineWidth=[0.0049 0.0076])`

Data Types: double

ResistorLength — Length of resistor in meters

0.0020 (default) | positive scalar

Length of the resistor in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitterUnequal(ResistorSLength=0.0050)`

Data Types: double

Resistance — Resistance value

106 (default) | positive scalar

Resistance value in ohms, specified as a positive scalar.

Example: `splitter = wilkinsonSplitterUnequal(Resistance=50)`

Data Types: double

Height — Height of Wilkinson splitter from ground plane

0.0016 (default) | positive scalar

Height of the Wilkinson splitter from the ground plane in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitterUnequal(Height=0.0076)`

Data Types: double

GroundPlaneWidth — Width of ground plane in meters

0.0300 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitterUnequal(GroundPlaneWidth=0.046)`

Example: double

Substrate — Type of dielectric material

'Teflon' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. The thickness of the default dielectric material Teflon is 0.0016 m or the same value as the Height property.

Example: `d = dielectric("FR4"); splitter = wilkinsonSplitterUnequal(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'Copper' (default) | metal object

Type of metal used in the conducting layers, specified as a metal object.

Example: `m = metal("PEC"); splitter = wilkinsonSplitterUnequal(Conductor=m)`

Data Types: string | char

Object Functions

charge	Calculate and plot charge distribution
current	Calculate and plot current distribution
design	Design unequal Wilkinson splitter around specified frequency
feedCurrent	Calculate current at feed port
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component

shapes Extract all metal layer shapes of PCB component
show Display PCB component structure or PCB shape
sparameters Calculate S-parameters for RF PCB objects

Examples

Create Default Unequal Wilkinson Splitter

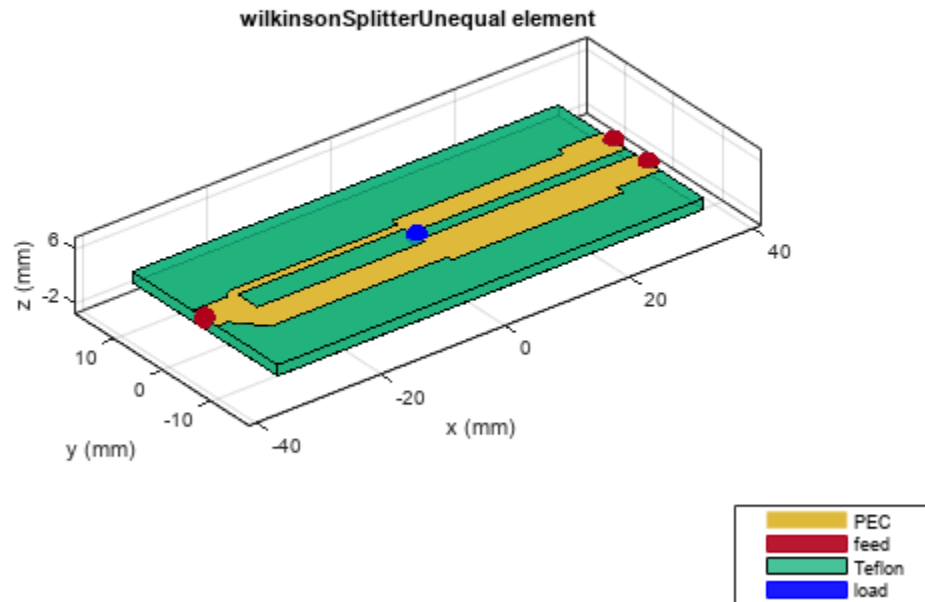
Create and view a default unequal Wilkinson splitter.

```
splitter = wilkinsonSplitterUnequal

splitter =
  wilkinsonSplitterUnequal with properties:

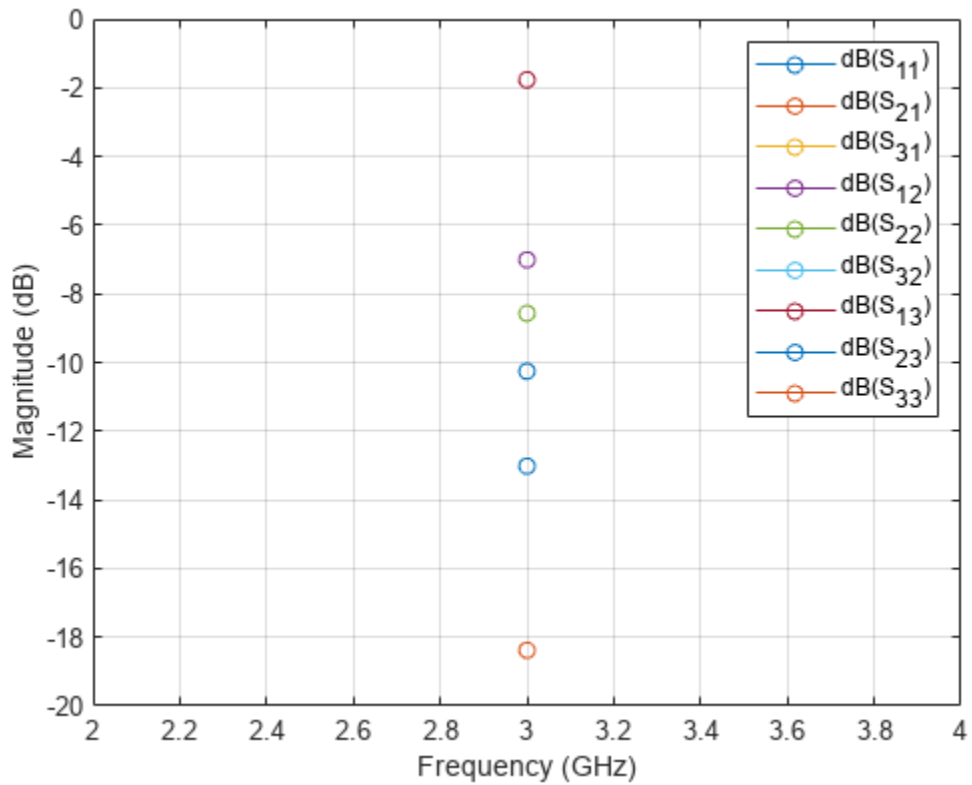
    PortLineLength: 0.0070
    PortLineWidth: 0.0051
    SplitLineLength: 0.0279
    SplitLineWidth: [0.0014 0.0049]
    MatchLineLength: 0.0277
    MatchLineWidth: [0.0039 0.0066]
    ResistorLength: 0.0020
    Resistance: 106
    Height: 0.0016
    GroundPlaneWidth: 0.0300
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]

show(splitter)
```



Calculate and plot the S-parameters of the splitter at 3 GHz.

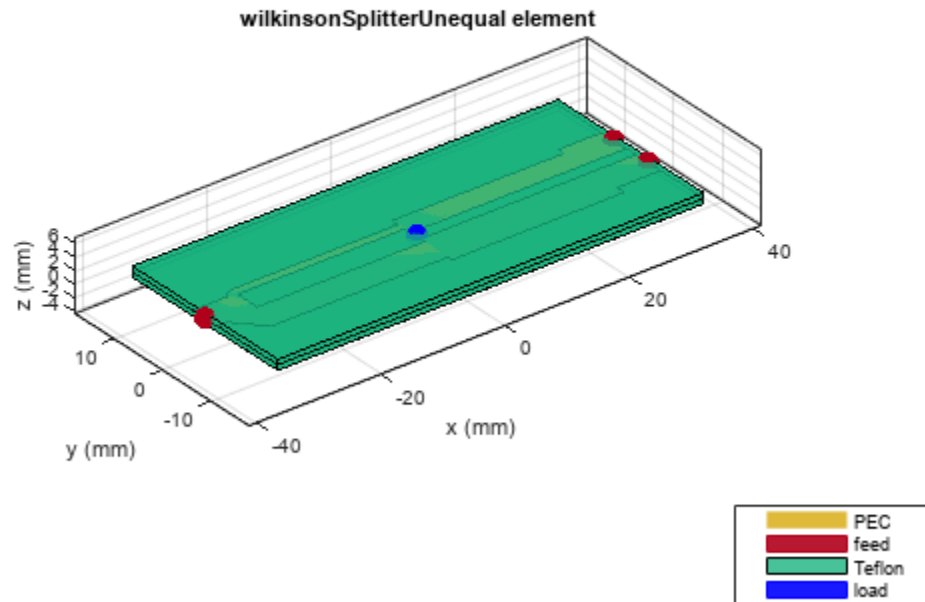
```
spar=sparameters(splitter,3e9);  
figure  
rfplot(spar);
```



Create Multilayer Unequal Wilkinson Splitter

Create and view a multilayer unequal Wilkinson splitter.

```
sub = dielectric(Name=["Teflon","Teflon"],EpsilonR=[2.1 2.1], ...
    LossTangent=[0 0],Thickness=[0.8e-3 0.8e-3]);
unsplitter = wilkinsonSplitterUnequal(Height=0.8e-3,Substrate=sub);
show(unsplitter)
```



Plot the charge and current on this splitter at 3 GHz.

```
figure  
charge(unsplitter,3e9)
```

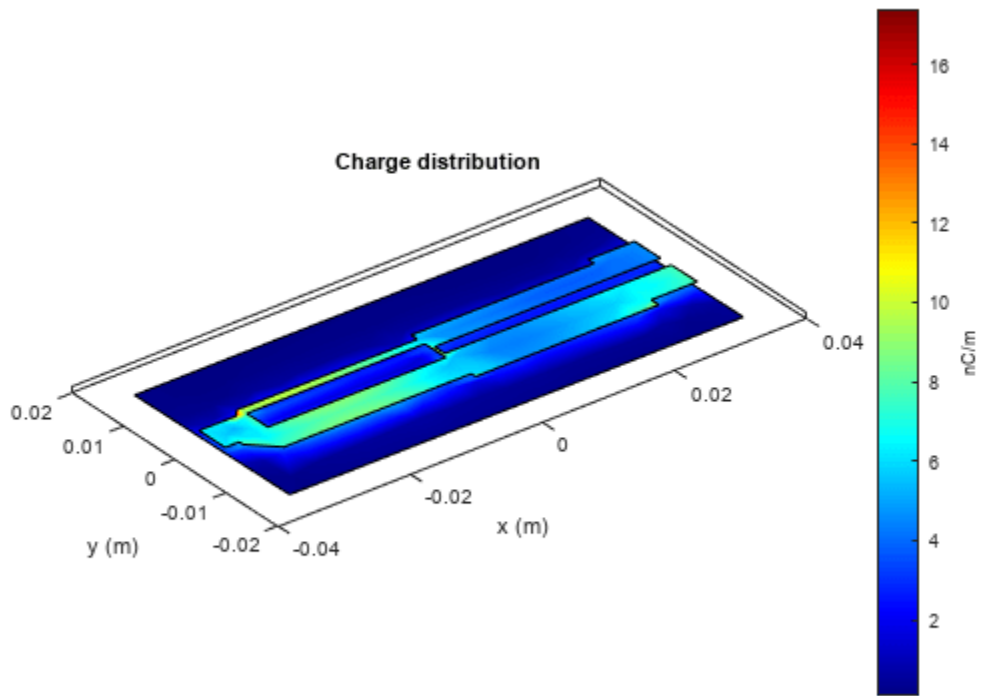
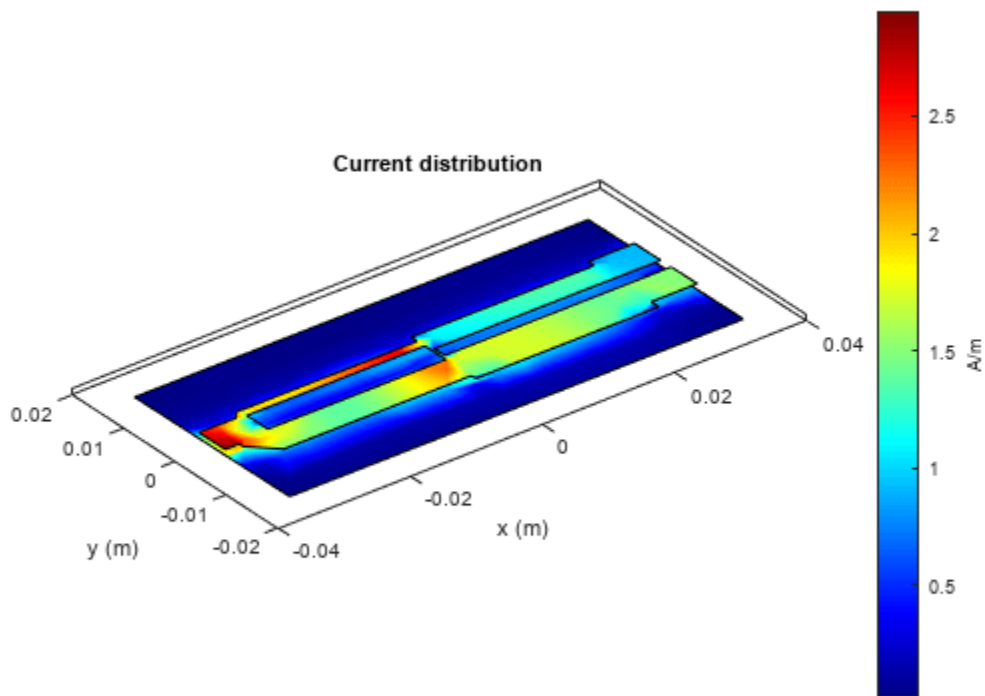


figure
current(unsplitter,3e9)



Version History

Introduced in R2021b

Behavioral Analysis for Unequal Wilkinson Power Splitter

Use the `sparameters` function and the `pcbElement` object to perform the behavioral analysis of a unequal Wilkinson power splitter.

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

`wilkinsonSplitter`

Topics

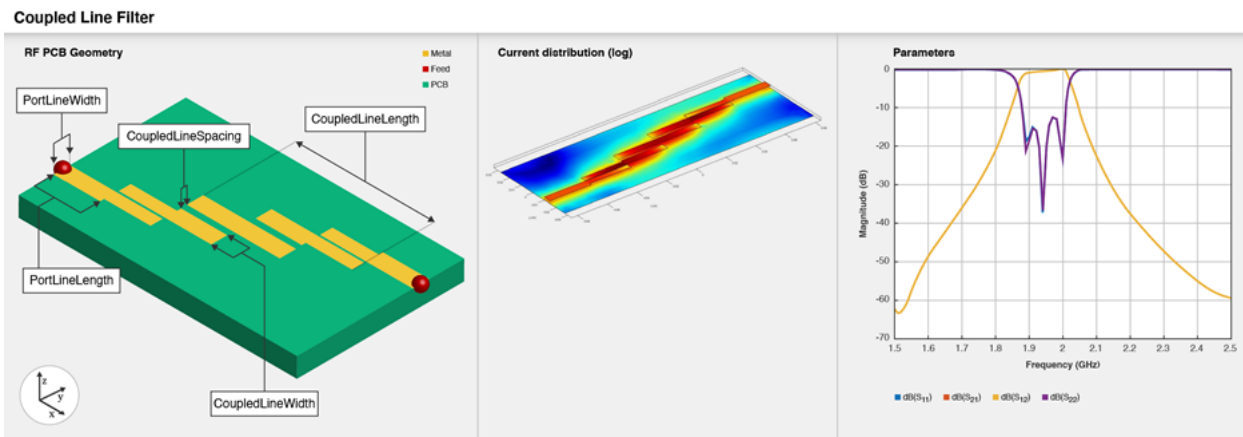
“Behavioral Models”

filterCoupledLine

Create coupled line filter in microstrip form

Description

Use the `filterCoupledLine` object to create a coupled line filter in microstrip form. The filter structure consists of open-circuited coupled microstrip lines. You can control the bandwidth of the filter by varying the filter order, width, and distance between the coupled lines.



Creation

Syntax

```
filter = filterCoupledLine
filter = filterCoupledLine(Name=Value)
```

Description

`filter = filterCoupledLine` creates a default coupled line filter using a Teflon substrate. The default passband of the filter is centered around 2 GHz.

`filter = filterCoupledLine(Name=Value)` sets “Properties” on page 1-125 using one or more name-value arguments. For example, `filterCoupledLine(FilterOrder=5)` creates a fifth-order coupled line filter. Properties not specified retain their default values.

Properties

FilterOrder – Filter order

3 (default) | positive scalar

Filter order, specified as a positive scalar.

Example: `filter = filterCoupledLine(FilterOrder=5)`

Data Types: double

PortLineLength — Length of input and output lines

0.0279 (default) | positive scalar

Length of the input and output lines in meters, specified as a positive scalar.

Example: `filter = filterCoupledLine(PortLineLength=0.0553)`

Data Types: double

PortLineWidth — Width of input and output lines

0.0051 (default) | positive scalar

Width of the input and output lines in meters, specified as a positive scalar.

Example: `filter = filterCoupledLine(PortLineWidth=0.0087)`

Data Types: double

CoupledLineLength — Lengths of coupled lines

[0.0279 0.0279 0.0279 0.0279] (default) | vector

Lengths of the coupled lines in meters, specified as a vector of positive elements.

Example: `filter = filterCoupledLine(CoupledLineLength=[0.0553 0.0553 0.0553 0.0553])`

Data Types: double

CoupledLineWidth — Widths of coupled lines

[0.0036 0.0049 0.0049 0.0036] (default) | vector

Widths of the coupled lines in meters, specified as a vector of positive elements.

Example: `filter = filterCoupledLine(CoupledLineWidth=[0.0046 0.0059 0.0059 0.0046])`

Data Types: double

CoupledLineSpacing — Distance between coupled lines

[1.8270e-04 0.0019 0.0019 1.8270e-04] (default) | vector

Distance between the coupled lines in meters, specified as a vector of positive elements.

Example: `filter = filterCoupledLine(CoupledLineSpacing=[2.8270e-04 0.0020 0.0020 2.8270e-04])`

Data Types: double

Height — Height of coupled line filter from ground plane

0.0016 (default) | positive scalar

Height of the coupled line filter from the ground plane in meters, specified as a positive scalar. For multilayer dielectrics, use the `Height` property to create the filter between the two dielectric layers.

Example: `filter = filterCoupledLine(Height=0.0028)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0551 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `filter = filterCoupledLine(GroundPlaneWidth=0.0048)`

Data Types: double

Substrate — Type of dielectric material

'Teflon' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. The thickness of the default dielectric material Teflon is 0.0016 m.

Example: `d = dielectric("FR4"); filter = filterCoupledLine(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'PEC' (default) | metal object

Type of metal used in the conducting layers, specified as a metal object.

Example: `m = metal("Copper"); filter = filterCoupledLine(Conductor=m)`

Data Types: string | char

Object Functions

charge	Calculate and plot charge distribution
current	Calculate and plot current distribution
design	Design coupled line filter around specified frequency
feedCurrent	Calculate current at feed port
getZ0	Calculate characteristic impedance of transmission line
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples**Create Default Coupled Line Filter**

Create and view a default coupled line filter.

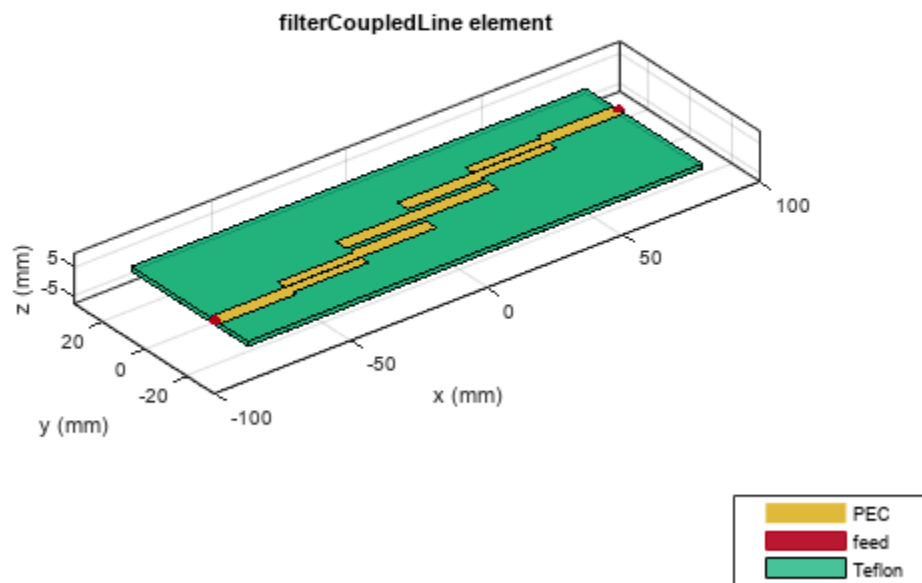
`coupledfilter = filterCoupledLine`

```
coupledfilter =
  filterCoupledLine with properties:
```

```
    FilterOrder: 3
    PortLineLength: 0.0279
    PortLineWidth: 0.0051
    CoupledLineLength: [0.0279 0.0279 0.0279 0.0279]
    CoupledLineWidth: [0.0036 0.0049 0.0049 0.0036]
```

```
CoupledLineSpacing: [1.8270e-04 0.0019 0.0019 1.8270e-04]
      Height: 0.0016
GroundPlaneWidth: 0.0551
      Substrate: [1x1 dielectric]
      Conductor: [1x1 metal]
```

```
show(coupledfilter)
```



Coupled Line Filter at Specified Frequency

Create and view a coupled line filter at 3 GHz.

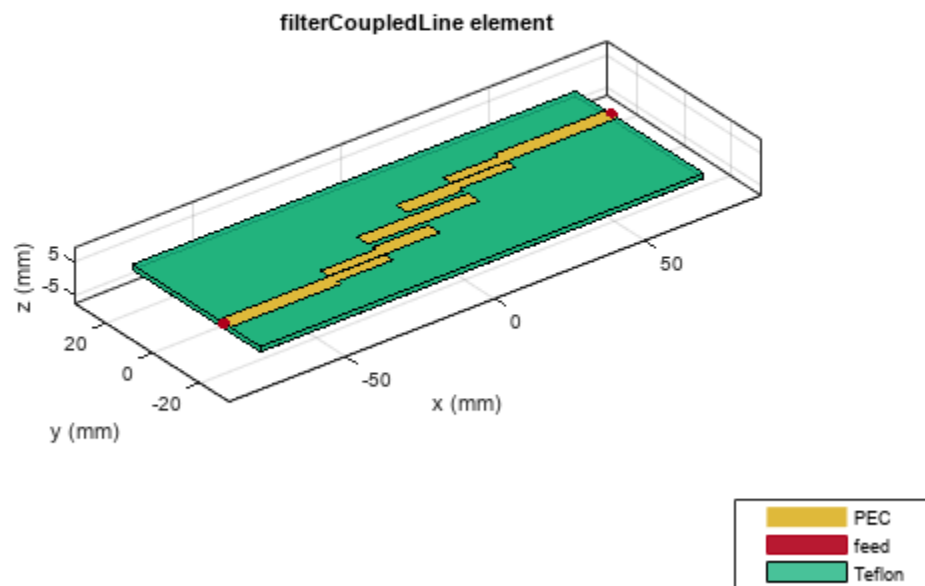
```
coupledfilter = design(filterCoupledLine,3e9)
```

```
coupledfilter =  
  filterCoupledLine with properties:
```

```
      FilterOrder: 3  
      PortLineLength: 0.0372  
      PortLineWidth: 0.0051  
      CoupledLineLength: [0.0186 0.0186 0.0186 0.0186]  
      CoupledLineWidth: [0.0036 0.0049 0.0049 0.0036]  
      CoupledLineSpacing: [1.8270e-04 0.0019 0.0019 1.8270e-04]  
      Height: 0.0016
```

```
GroundPlaneWidth: 0.0551  
Substrate: [1x1 dielectric]  
Conductor: [1x1 metal]
```

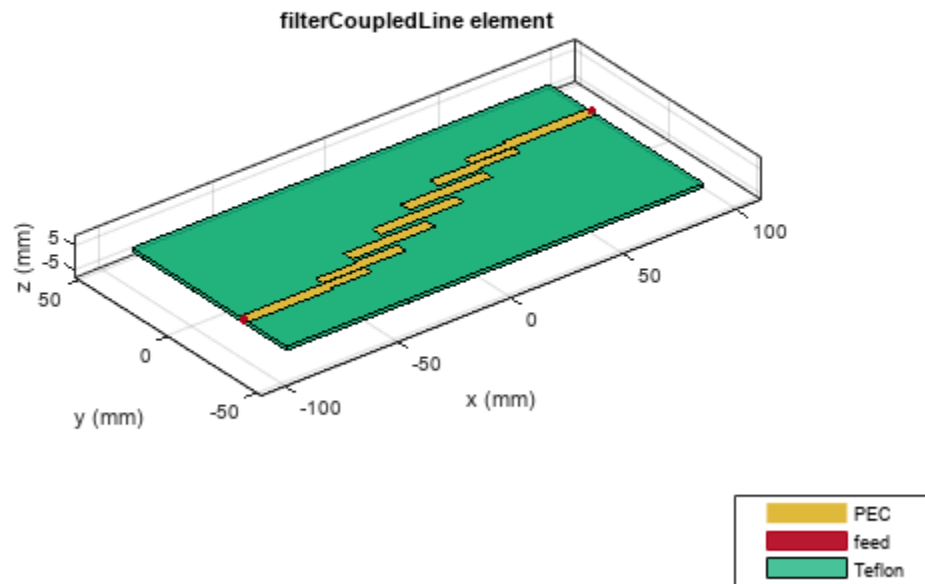
```
show(coupledfilter)
```



Fifth-Order Coupled Line Chebyshev Filter

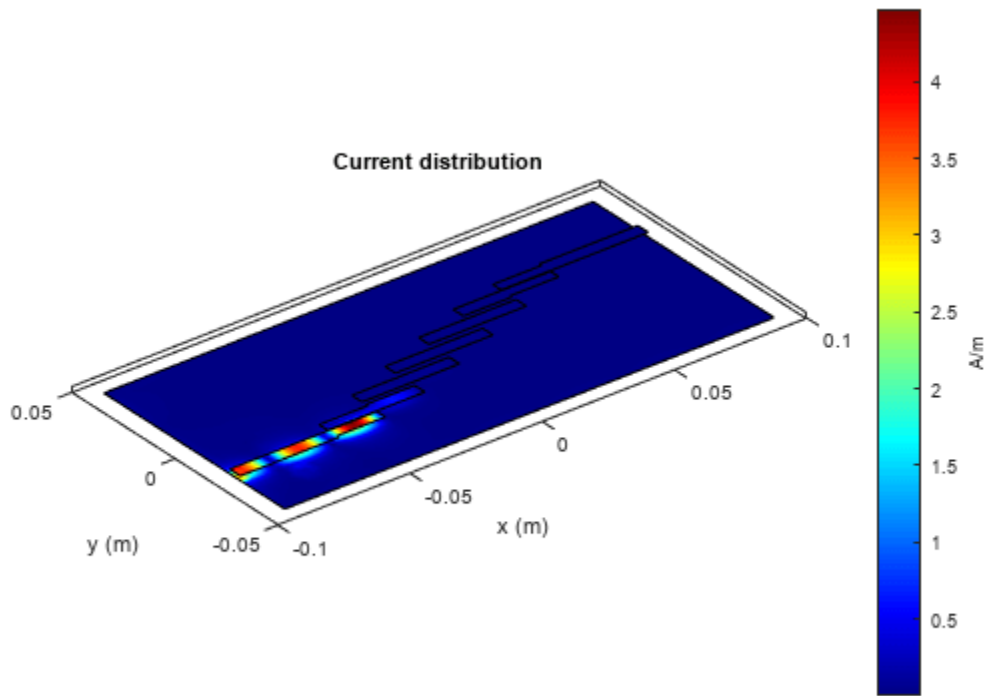
Design and view a fifth-order coupled-line Chebyshev filter at 3 GHz with a ripple factor of 0.5 dB.

```
coupledfilter = filterCoupledLine(FilterOrder=5);  
coupledfilter = design(coupledfilter,3e9,FilterType="Chebyshev",RippleFactor=0.5);  
show(coupledfilter)
```

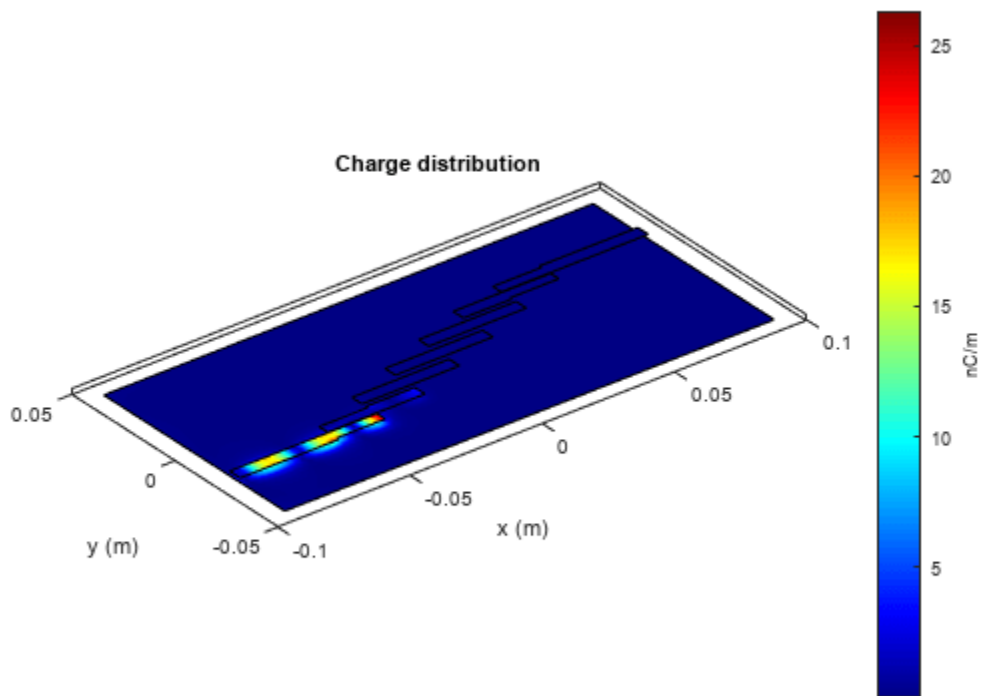


Plot the current and charge distribution of this filter at 5 GHz.

```
figure  
current(coupledfilter,5e9)
```



```
figure  
charge(coupledfilter,5e9)
```



Version History

Introduced in R2021b

References

- [1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.
- [2] Ragani, Taoufik, N. Amar Touhami, and M. Agoutane. "Designing a Microstrip Coupled Line Bandpass Filter." *International Journal of Engineering & Technology* 2, no. 4 (September 6, 2013): 266. <https://doi.org/10.14419/ijet.v2i4.1173>.

See Also

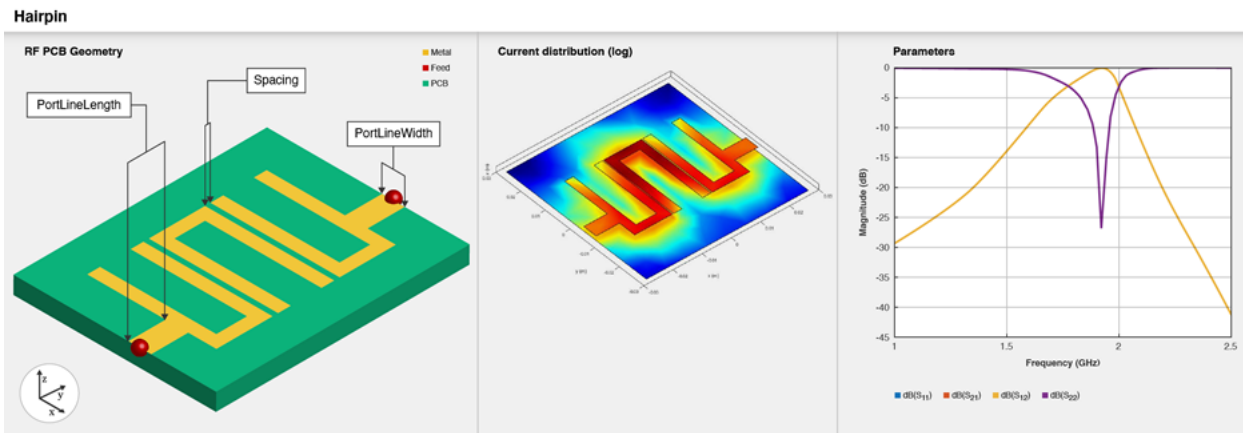
`filterHairpin` | `filterStepImpedanceLowPass`

filterHairpin

Create hairpin filter in microstrip form

Description

Use the `filterHairpin` object to create a hairpin filter in microstrip form.



Hairpin filters are easy to fabricate. Design these filters at different frequencies by changing the roll-offs and the ripple factors. You can control the bandwidth of the filter by varying the filter order, width, and the distance between the coupled lines. There are two types of hairpin filters based on the feed: tapped-line input hairpin filters and coupled-line input hairpin filters.

Creation

Syntax

```
filter = filterHairpin
filter = filterHairpin(Name=Value)
```

Description

`filter = filterHairpin` creates a default hairpin filter using a Teflon substrate. The default passband of the filter is 2 GHz.

`filter = filterHairpin(Name=Value)` sets "Properties" on page 1-134 using one or more name-value arguments. For example, `filterHairpin(Resonator=ubendMitered)` creates a hairpin filter with a mitered u-bend element as the resonator. Properties not specified retain their default values

Properties

Resonator — Shape of hairpin element

uBendRightAngle (default) | ubendMitered | ubendCurved

Shape of the hairpin element, specified as either uBendRightAngle, ubendMitered, or ubendCurved.

Example: filter = filterHairpin(Resonator=ubendCurved)

Data Types: char | string

FilterOrder — Filter order

3 (default) | positive scalar

Filter order, specified as a positive scalar.

Example: filter = filterHairpin(FilterOrder=5)

Data Types: double

ResonatorOffset — Y-offset of each resonator

[0 0 0] (default) | vector

Y-offset of each resonator in meters, specified as a vector.

Example: filter = filterHairpin(ResonatorOffset=[0.02 0.02 0.02])

Data Types: double

Spacing — Distance between hairpin bends

[4.0000e-04 4.0000e-04] (default) | vector

Distance between the hairpin bends in meters, specified as a vector of positive elements.

Example: filter = filterHairpin(Spacing=[0.02 0.02 0.02])

Data Types: double

PortLineLength — Length of input and output lines

0.0080 (default) | positive scalar

Length of the input and output lines in meters, specified as a positive scalar.

Example: filter = filterHairpin(PortLineLength=0.0553)

Data Types: double

PortLineWidth — Width of input and output lines

0.0050 (default) | positive scalar

Width of the input and output lines in meters, specified as a positive scalar.

Example: filter = filterHairpin(PortLineWidth=0.0087)

Data Types: double

FeedOffset — Y-offset for input and output lines

[-0.0055 -0.0055] (default) | vector

Y-offset for the input and the output lines, specified as a vector.

Example: `filter = filterHairpin(FeedOffset=[-0.002 -0.002])`

Data Types: double

FeedType — Type of feed at input and output ports

"Tapped" (default) | "Coupled"

Type of feed at the input and output ports, specified as either "Tapped" or "Coupled".

Example: `filter = filterHairpin(FeedType="Coupled")`

Data Types: char | string

CoupledLineLength — Length of coupled feed lines

[0.0279 0.0279 0.0279 0.0279] (default) | vector

Length of the coupled feed lines in meters, specified as a vector of positive elements.

Example: `filter = filterHairpin(CoupledLineLength=[0.0553 0.0553 0.0553 0.0553])`

Dependencies

To enable `CoupledLineLength`, set the `FeedType` property to "Coupled".

Data Types: double

CoupledLineWidth — Width of coupled feed lines

[0.0036 0.0049 0.0049 0.0036] (default) | vector

Width of the coupled feed lines in meters, specified as a vector of positive elements.

Example: `filter = filterHairpin(CoupledLineWidth=[0.0046 0.0059 0.0059 0.0046])`

Dependencies

To enable `CoupledLineLength`, set the `FeedType` property to "Coupled".

Data Types: double

CoupledLineSpacing — Distance between feed line and hairpin

[1.8270e-04 0.0019 0.0019 1.8270e-04] (default) | vector

Distance between the feed line and the hairpin in meters, specified as a vector.

Example: `filter = filterHairpin(CoupledLineSpacing=[2.8270e-04 0.0020 0.0020 2.8270e-04])`

Dependencies

To enable `CoupledLineLength`, set the `FeedType` property to "Coupled".

Data Types: double

Height — Height of hairpin filter from ground plane

0.0016 (default) | positive scalar

Height of the hairpin filter from the ground plane in meters, specified as a positive scalar. In the case of a multilayer substrate, you can use the `Height` property to create a hairpin filter where the two dielectrics interface.

Example: `filter = filterHairpin(Height=0.020)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0567 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `filter = filterHairpin(GroundPlaneWidth=[0.0679])`

Data Types: double

Substrate — Type of dielectric material

'Teflon' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. The thickness of the default dielectric material Teflon is 0.0016 m or the same value as the Height property.

Example: `d = dielectric("FR4"); filter = filterHairpin(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'PEC' (default) | metal object

Type of metal used in the conducting layers, specified as a metal object.

Example: `m = metal("Copper"); filter = filterHairpin(Conductor=m)`

Data Types: string | char

Object Functions

charge	Calculate and plot charge distribution
current	Calculate and plot current distribution
design	Design hairpin filter around specified frequency
feedCurrent	Calculate current at feed port
getZ0	Calculate characteristic impedance of transmission line
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples

Create Default Hairpin Filter

Create and view a default hairpin filter.

```
hairpinfilter = filterHairpin
```

```
hairpinfilter =  
    filterHairpin with properties:
```

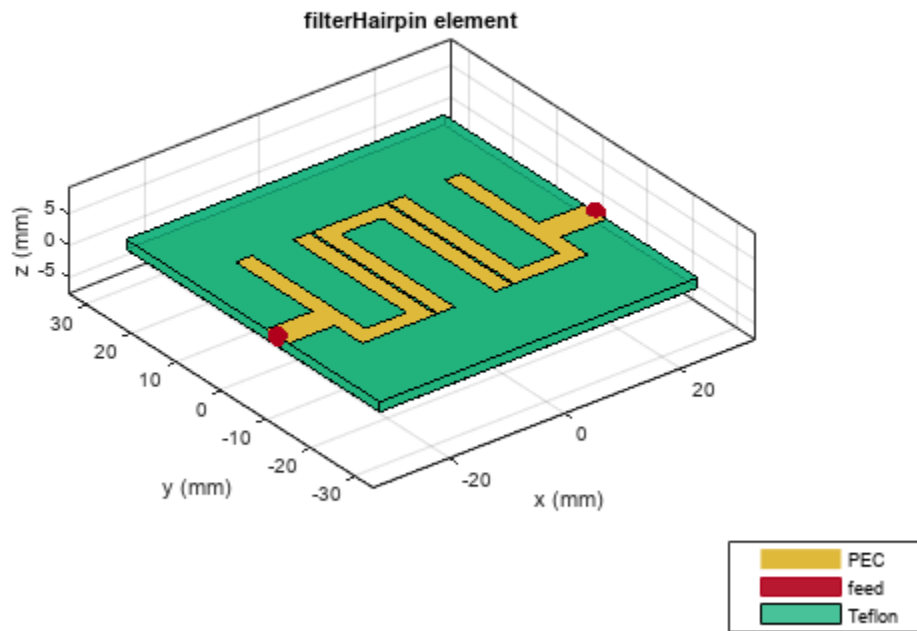
```
        Resonator: [1x1 ubendRightAngle]
```

```

FilterOrder: 3
ResonatorOffset: [0 0 0]
  Spacing: [4.0000e-04 4.0000e-04]
PortLineLength: 0.0080
PortLineWidth: 0.0050
  FeedOffset: [-0.0055 -0.0055]
  FeedType: 'Tapped'
  Height: 0.0016
GroundPlaneWidth: 0.0567
  Substrate: [1x1 dielectric]
  Conductor: [1x1 metal]

```

```
show(hairpinfilter)
```



Fifth-Order Coupled Hairpin Filter

Create and view a fifth-order coupled hairpin filter.

```
hairpinfilter = filterHairpin(FeedType="Coupled")
```

```
hairpinfilter =
  filterHairpin with properties:
```

```
    Resonator: [1x1 ubendRightAngle]
```

```
    FilterOrder: 3
  ResonatorOffset: [0 0 0]
    Spacing: [4.0000e-04 4.0000e-04]
    FeedOffset: [-0.0055 -0.0055]
  PortLineLength: 0.0080
  PortLineWidth: 0.0050
    FeedType: 'Coupled'
  CoupledLineLength: 0.0279
  CoupledLineWidth: 0.0029
  CoupledLineSpacing: [1.8270e-04 1.8270e-04]
    Height: 0.0016
  GroundPlaneWidth: 0.0567
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

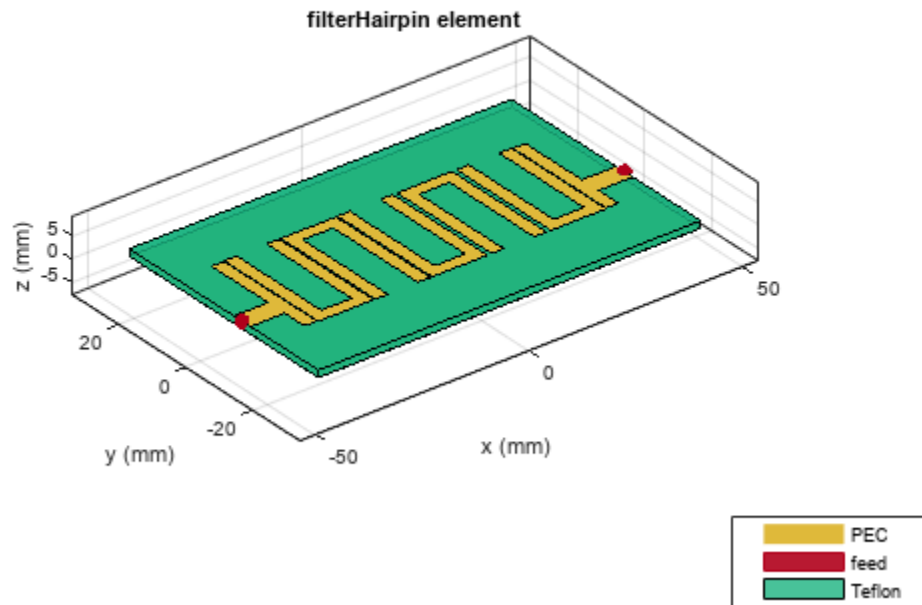
```
hairpinfilter.FilterOrder = 5
```

```
hairpinfilter =
```

```
  filterHairpin with properties:
```

```
    Resonator: [1x1 ubendRightAngle]
  FilterOrder: 5
  ResonatorOffset: [0 0 0 0 0]
    Spacing: [4.0000e-04 4.0000e-04 1.0000e-03 1.0000e-03]
    FeedOffset: [-0.0055 -0.0055]
  PortLineLength: 0.0080
  PortLineWidth: 0.0050
    FeedType: 'Coupled'
  CoupledLineLength: 0.0279
  CoupledLineWidth: 0.0029
  CoupledLineSpacing: [1.8270e-04 1.8270e-04]
    Height: 0.0016
  GroundPlaneWidth: 0.0567
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

```
show(hairpinfilter)
```



Plot the current and charge distribution of the filter at 2 GHz.

```
figure  
current(hairpinfilter,2e9)
```

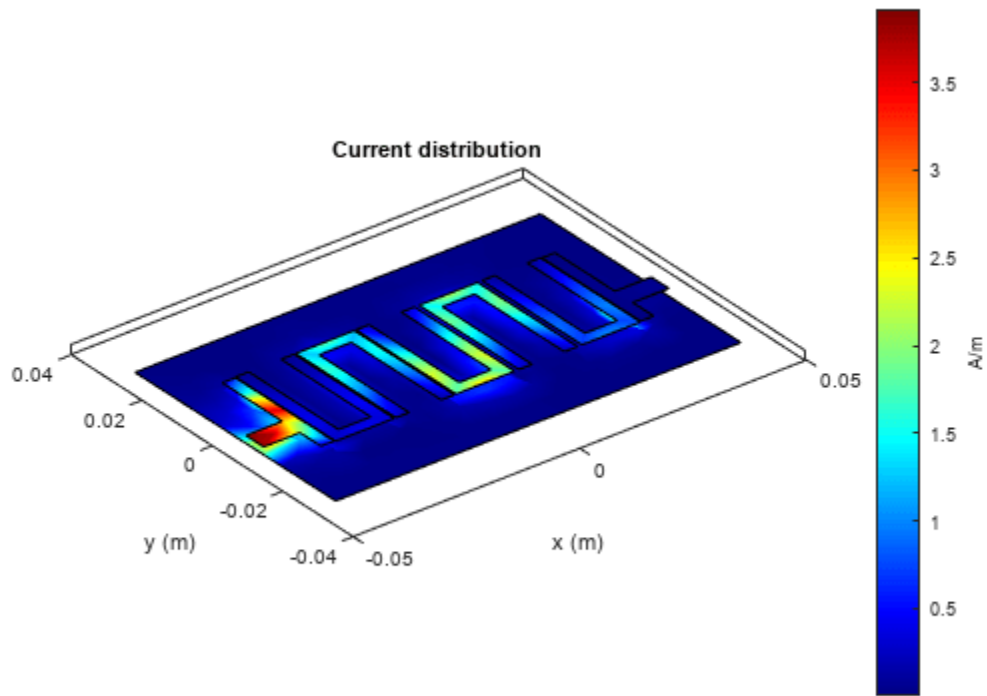
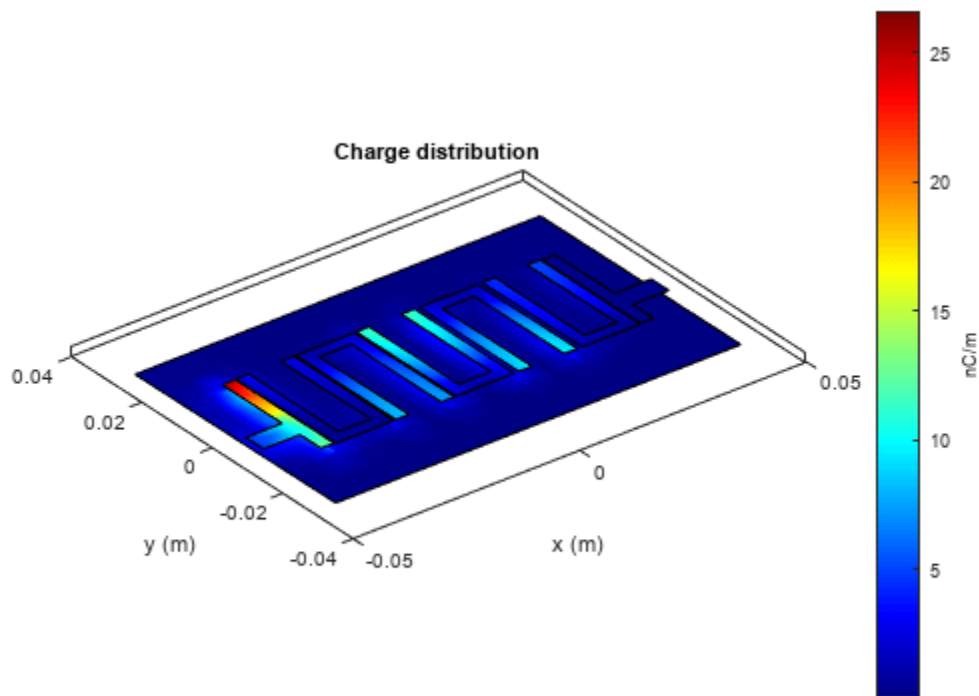


figure
charge(hairpinfilter,2e9)



Version History

Introduced in R2021b

References

- [1] Bankey, Kavita, and Abhinav Bhargava. "Design of Compact Microstrip Hairpin Multi Bandpass Filter." *International Journal of Scientific Progress and Research* 34, no. 96 (2017): 66–69.
- [2] Parikh, Nikunj, Pragma Katare, Ketan Kathal, Nandini Patel, and Gaurav Chaitanya. "Design and Analysis of Hairpin Micro-Strip Line Band Pass Filter." *International Journal of Innovative Research in Electrical, Electronics, Instrumentation, and Control Engineering* 3 (April 2015). <https://doi.org/10.17148/IJIREECE.2015.3512>.

See Also

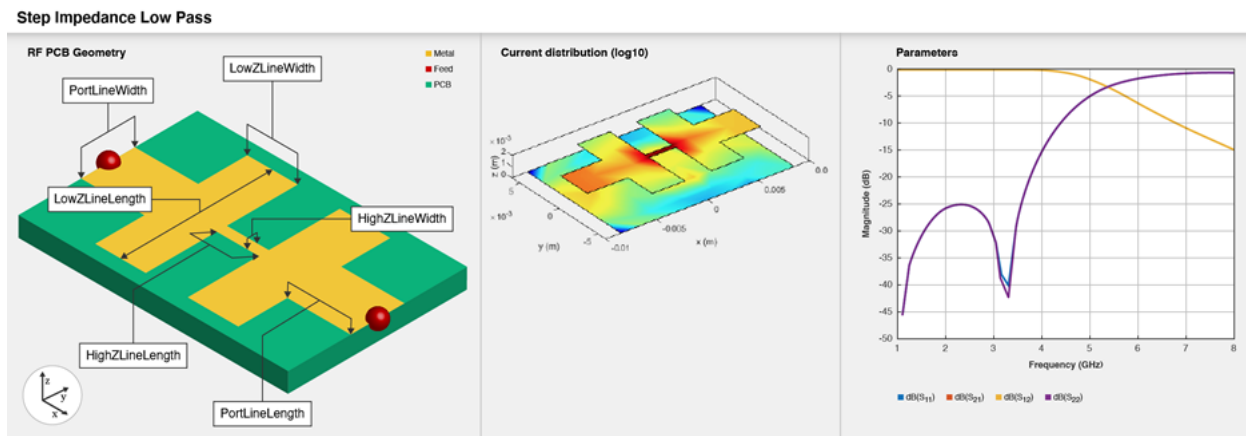
filterCoupledLine | filterStepImpedanceLowPass

filterStepImpedanceLowPass

Create stepped impedance lowpass filter in microstrip form

Description

Use the `filterStepImpedanceLowPass` object to create a stepped impedance lowpass filter in microstrip form.



The stepped impedance lowpass microstrip filters have a cascaded structure of alternating high- and low-impedance transmission lines. These lines are considerably shorter in length than the design wavelength and act as semi-lumped elements. The high-impedance lines act as series inductors, and the low-impedance lines act as shunt capacitors. This filter structure realizes a Pi LC ladder type of a lowpass filter. You can control the impedance by adjusting the width of the strip. This filter is used in radar, satellite, and terrestrial communications and in electronic counter-measure applications.

Creation

Syntax

```
filter = filterStepImpedanceLowPass
filter = filterStepImpedanceLowPass(Name=Value)
```

Description

`filter = filterStepImpedanceLowPass` creates a default stepped impedance lowpass filter. The default filter dimensions are for a cutoff frequency 5 GHz.

`filter = filterStepImpedanceLowPass(Name=Value)` sets "Properties" on page 1-143 using one or more name-value arguments. For example, `filterStepImpedanceLowPass(FilterOrder=10)` creates a tenth-order stepped impedance lowpass filter. Properties not specified retain their default values

Properties

FilterOrder — Filter order

3 (default) | positive scalar in the range [3,11]

Filter order, specified as a positive scalar. The minimum filter order you can specify is 3 and the maximum order is 11.

Example: `filter = filterStepImpedanceLowPass(FilterOrder=5)`

Data Types: double

PortLineLength — Length of input and output lines

0.0034 (default) | positive scalar

Length of the input and output lines in meters, specified as a positive scalar.

Example: `filter = filterStepImpedanceLowPass(PortLineLength=0.0553)`

Data Types: double

PortLineWidth — Width of input and output lines

0.0040 (default) | positive scalar

Width of the input and output lines in meters, specified as a positive scalar.

Example: `filter = filterStepImpedanceLowPass(PortLineWidth=0.0087)`

Data Types: double

LowZLineWidth — Width of low-impedance line

0.0096 (default) | positive scalar

Width of the low-impedance line in meters, specified as a positive scalar.

Example: `filter = filterStepImpedanceLowPass(LowZLineWidth=0.0553)`

Data Types: double

HighZLineWidth — Width of high-impedance line

5.0000e-04 (default) | positive scalar

Width of the high-impedance line in meters, specified as a positive scalar.

Example: `filter = filterStepImpedanceLowPass(HighZLineWidth=0.0553)`

Data Types: double

LowZLineLength — Length of low-impedance line

0.0032 (default) | positive scalar

Length of the low-impedance line in meters, specified as a positive scalar.

Example: `filter = filterStepImpedanceLowPass(LowZLineLength=0.0553)`

Data Types: double

HighZLineLength — Length of high-impedance line

0.0026 (default) | positive scalar

Length of the high-impedance line in meters, specified as a positive scalar.

Example: `filter = filterStepImpedanceLowPass(HighZLineWidth=0.0553)`

Data Types: `double`

Height — Height of filter from ground plane

`0.0016` (default) | positive scalar

Height of the filter from the ground plane in meters, specified as a positive scalar.

Example: `filter = filterStepImpedanceLowPass(Height=0.020)`

Data Types: `double`

GroundPlaneWidth — Width of ground plane

`0.012` (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `filter = filterStepImpedanceLowPass(GroundPlaneWidth=0.013)`

Data Types: `double`

Substrate — Type of dielectric material

`dielectric` object

Type of dielectric material used as a substrate, specified as a `dielectric` object. The default value is a `dielectric` object with these properties:

- `Name`—{'CustomDielectric'}
- `EpsilonR`—3.7
- `LossTangent`—0.001
- `Thickness`—1.6e-3

Example: `d = dielectric("FR4"); filter = filterStepImpedanceLowPass(Substrate=d)`

Data Types: `string` | `char`

Conductor — Type of metal used in conducting layers

`'PEC'` (default) | `metal` object

Type of metal used in the conducting layers, specified as a `metal` object.

Example: `m = metal("Copper"); filter = filterStepImpedanceLowPass(Conductor=m)`

Data Types: `string` | `char`

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>design</code>	Design stepped impedance low pass filter around desired cut-off frequency
<code>feedCurrent</code>	Calculate current at feed port
<code>getZ0</code>	Calculate characteristic impedance of transmission line
<code>layout</code>	Plot all metal layers and board shape
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>shapes</code>	Extract all metal layer shapes of PCB component

show Display PCB component structure or PCB shape
sparameters Calculate S-parameters for RF PCB objects

Examples

Default Stepped Impedance Lowpass Filter

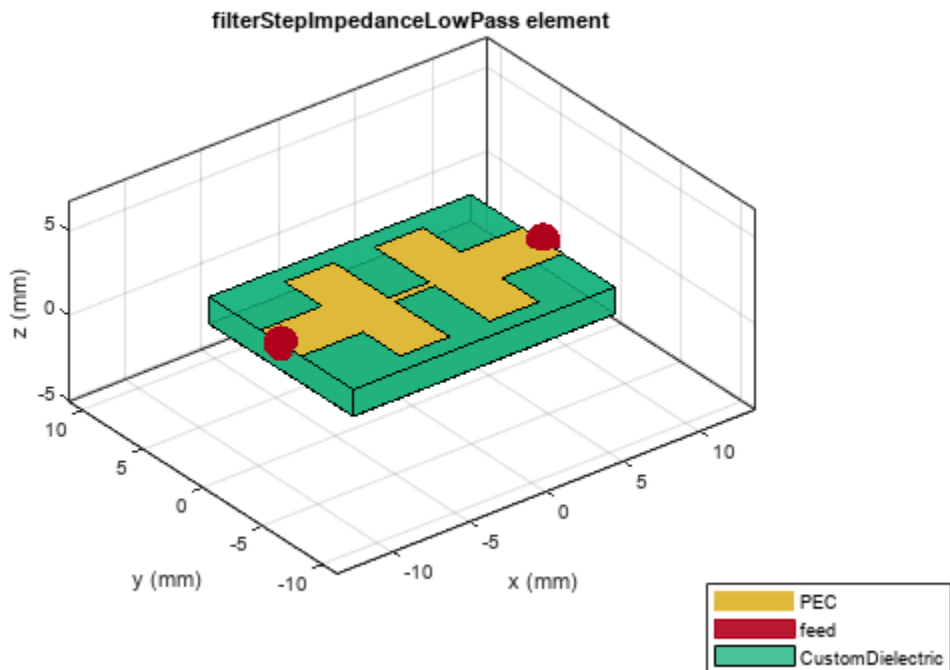
Create and view a default stepped impedance lowpass filter.

```
steppedfilter = filterStepImpedanceLowPass
```

```
steppedfilter =  
  filterStepImpedanceLowPass with properties:
```

```
    FilterOrder: 3  
    PortLineWidth: 0.0034  
    PortLineLength: 0.0040  
    HighZLineWidth: 5.0000e-04  
    LowZLineWidth: 0.0096  
    HighZLineLength: 0.0026  
    LowZLineLength: 0.0032  
    Height: 0.0016  
    GroundPlaneWidth: 0.0120  
    Substrate: [1x1 dielectric]  
    Conductor: [1x1 metal]
```

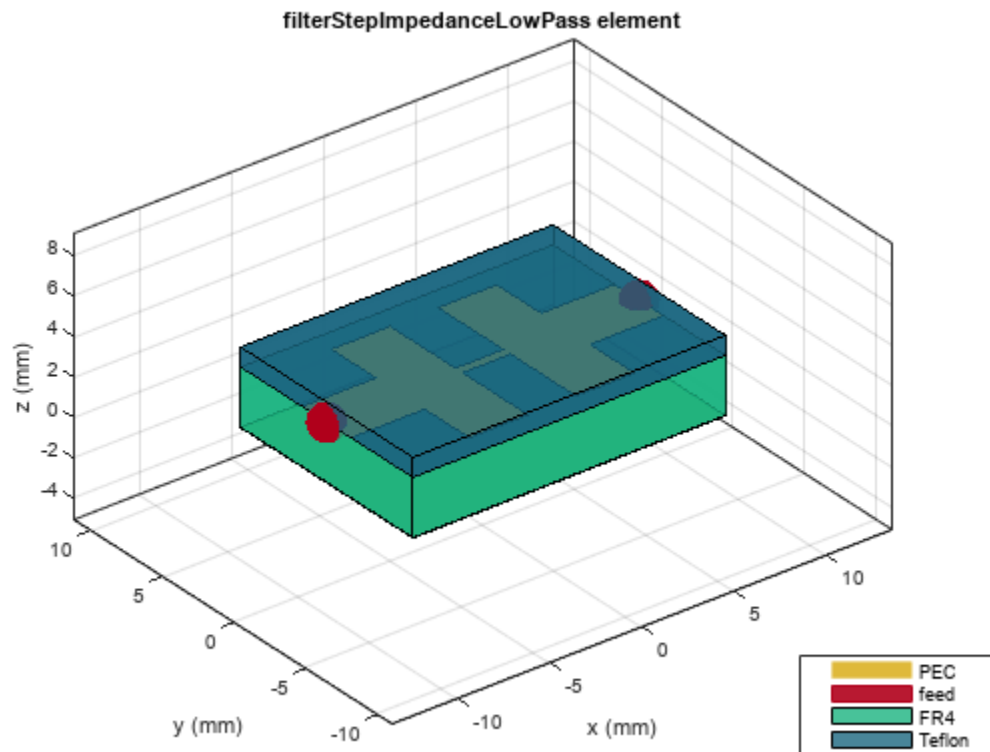
```
show(steppedfilter)
```



Create Stepped Impedance Lowpass Filter with Multilayer Dielectric Substrate

Create and view a stepped impedance lowpass filter with a multilayer dielectric substrate.

```
sub = dielectric("FR4", "Teflon");  
sub.Thickness = [0.003 0.001];  
steppedfilter = filterStepImpedanceLowPass;  
steppedfilter.Height = 0.003;  
steppedfilter.Substrate = sub;  
figure  
show(steppedfilter)
```



Plot the charge and current on the filter at 5 GHz.

```
figure  
charge(stepedfilter,5e9)
```

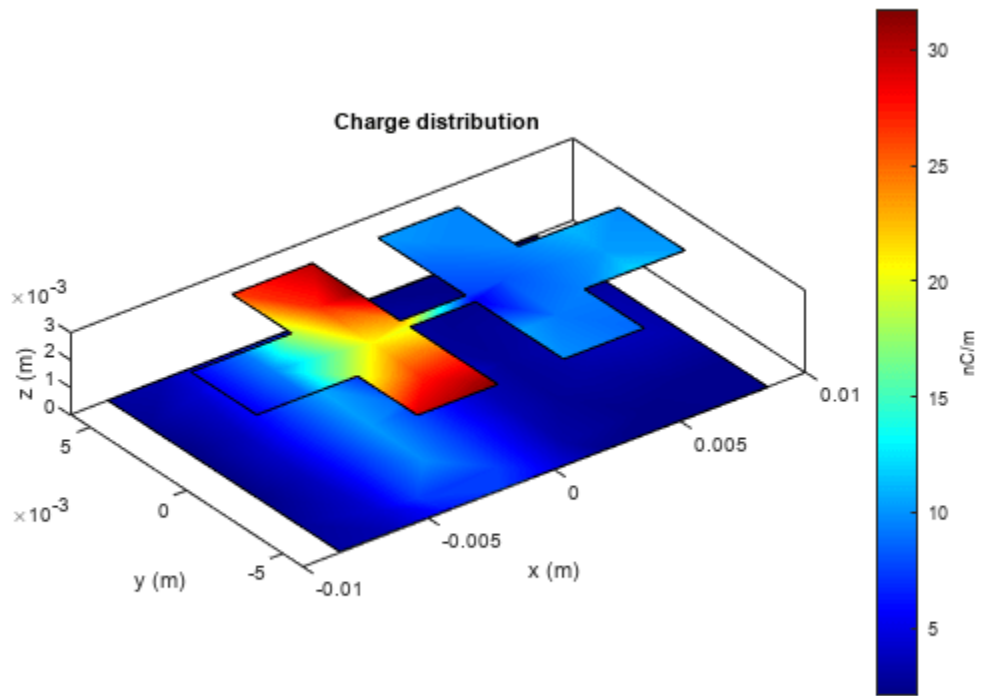
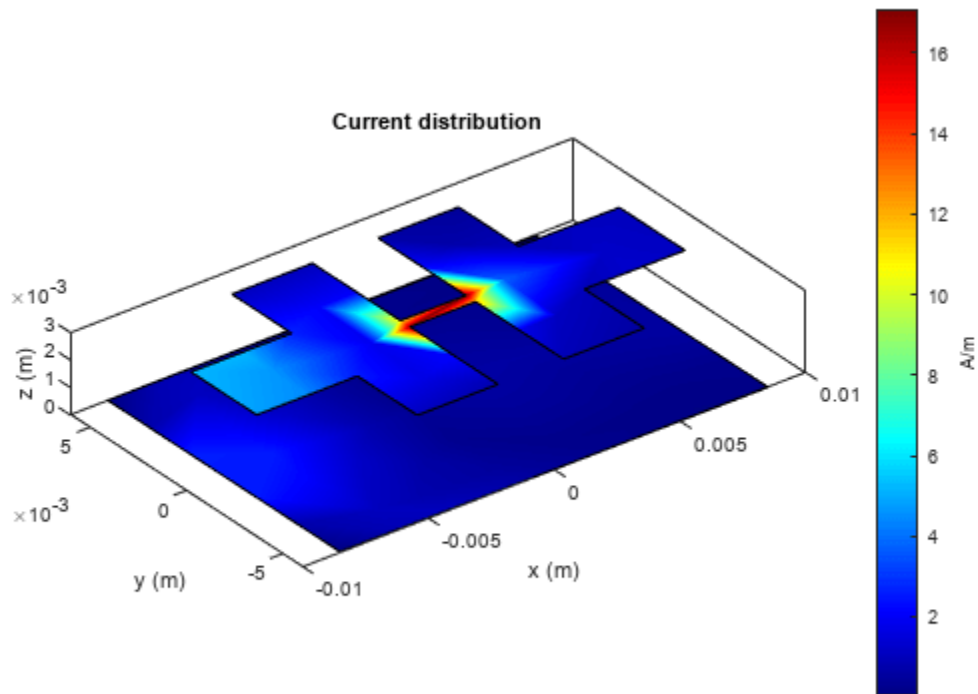


figure
current(stepedfilter,5e9)



```
info(stepedfilter)
```

```
ans = struct with fields:
    IsSolved: "true"
    IsMeshed: "true"
    MeshingMode: "auto"
    HasSubstrate: "true"
    HasLoad: "false"
    PortFrequency: []
    MemoryEstimate: "790 MB"
```

Version History

Introduced in R2021b

References

- [1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.
- [2] Garvansh, Abhay, Singh Kushwaha, Navita Singh, and Arun Kumar. "Implementation of Stepped Impedance Low Pass Microstrip Line Filter for Wireless Communication." *International Journal of Advanced Research in Computer and Communication Engineering* 3, no. 7 (July 2014): 7608–10.

- [3] Maity, Budhadeb. "Stepped Impedance Low Pass Filter Using Microstrip Line for C-Band Wireless Communication." In *2016 International Conference on Computer Communication and Informatics (ICCCI)*, 1-4. Coimbatore, India: IEEE, 2016. <https://doi.org/10.1109/ICCCI.2016.7480008>.

See Also

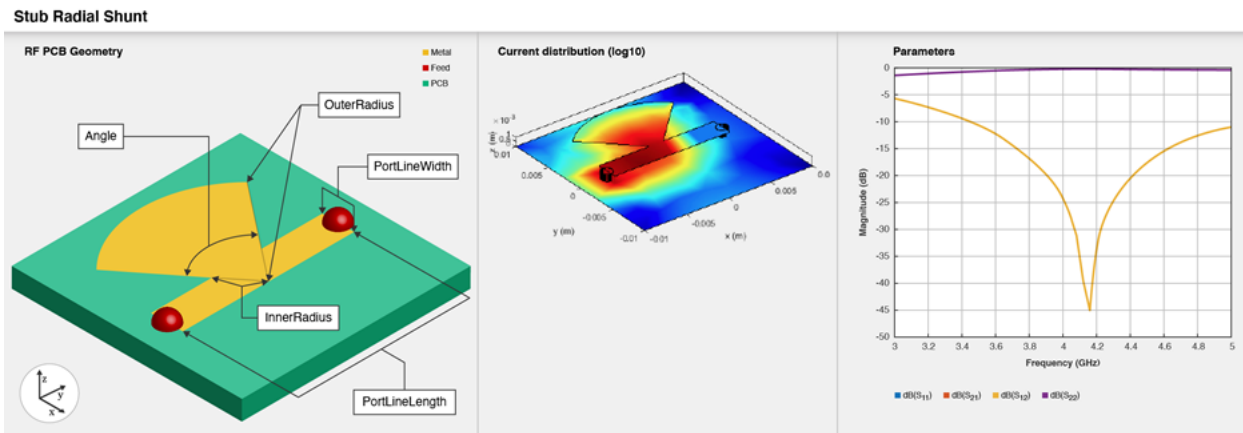
`filterCoupledLine` | `filterHairpin`

stubRadialShunt

Create single- and double-radial stub shunt on X-Y plane

Description

Use the `stubRadialShunt` object to create a single- or double-radial stub shunt on the X-Y plane.



Radial stubs provide broadly resonant RF short circuits by introducing the short at a concentrated point. When cascaded with high-impedance quarter-wavelength transmission lines, radial stubs provide an effectively decoupled network for microwave amplifiers and other active components.

Creation

Syntax

```
stub = stubRadialShunt
stub = stubRadialShunt(Name=Value)
```

Description

`stub = stubRadialShunt` creates a single-radial stub shunt in the X-Y plane. The stub dimensions are for the frequency range of 3-5 GHz with a resonant frequency of 4.2 GHz on the X-Y plane.

`stub = stubRadialShunt(Name=Value)` sets "Properties" on page 1-151 using one or more name-value arguments. For example, `stubRadialStub(OuterRadius=0.0070)` creates a radial stub shunt with an outer radius of 0.0070 meters. Properties not specified retain their default values.

Properties

StubType — Type of radial stub

"Single" (default) | "Double"

Type of radial stub, specified as "Single" or "Double".

Example: `stub = stubRadialShunt(StubType="Double")`

Data Types: string | char

OuterRadius — Outer radius of radial stub

0.0085 (default) | positive scalar | two-element vector

Outer radius of the radial stub in meters, specified as a positive scalar or a two-element vector of positive elements. Specify a two-element vector for a double-radial stub.

Example: `stub = stubRadialShunt(OuterRadius=0.0070)`

Data Types: double

InnerRadius — Inner radius of radial stub

0.0012 (default) | positive scalar | two-element vector

Inner radius of the radial stub in meters, specified as a positive scalar or a two-element vector of positive elements. Specify a two-element vector for a double-radial stub.

Example: `stub = stubRadialShunt(InnerRadius=0.0023)`

Data Types: double

Angle — Angle of stub

90 (default) | positive scalar in the range [5, 175] | two-element vector in the range [5, 175]

Angle of the stub in degrees, specified as a positive scalar or a two-element vector of positive elements. Specify a two-element vector for a double-radial stub. The stub angles must be greater than or equal to 5 degrees and less than or equal to 175 degrees.

Example: `stub = stubRadialShunt(Angle=60)`

Data Types: double

PortLineWidth — Width of microstrip line

0.0025 (default) | positive scalar

Width of the microstrip line in meters, specified as a positive scalar.

Example: `stub = stubRadialShunt(PortLineWidth=0.0035)`

Data Types: double

PortLineLength — Length of microstrip line

0.0137 (default) | positive scalar

Length of the microstrip line in meters, specified as a positive scalar.

Example: `stub = stubRadialShunt(PortLineLength=0.0237)`

Data Types: double

Height — Height of radial stub from ground plane

0.0016 (default) | positive scalar

Height of the radial stub from the ground plane, specified as a positive scalar.

Example: `stub = stubRadialShunt(Height=0.0015)`

Data Types: double

GroundPlaneLength — Length of ground plane

0.0200 (default) | positive scalar

Length of the ground plane in meters, specified as a positive scalar.

Example: `stub = stubRadialShunt(GroundPlaneLength=0.046)`

Example: double

GroundPlaneWidth — Width of ground plane

0.0200 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `stub = stubRadialShunt(GroundPlaneWidth=0.046)`

Example: double

Substrate — Type of dielectric material

'Teflon' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. The thickness of the default dielectric material Teflon is 0.8 mm or the same as the Height property.

Example: `d = dielectric("FR4"); stub = stubRadialShunt(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'PEC' (default) | metal object

Type of metal used in the conducting layers, specified as a metal object.

Example: `m = metal("PEC"); stub = stubRadialShunt(Conductor=m)`

Data Types: string | char

Object Functions

charge	Calculate and plot charge distribution
current	Calculate and plot current distribution
feedCurrent	Calculate current at feed port
getZ0	Calculate characteristic impedance of transmission line
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples

Create Default Radial Stub Shunt

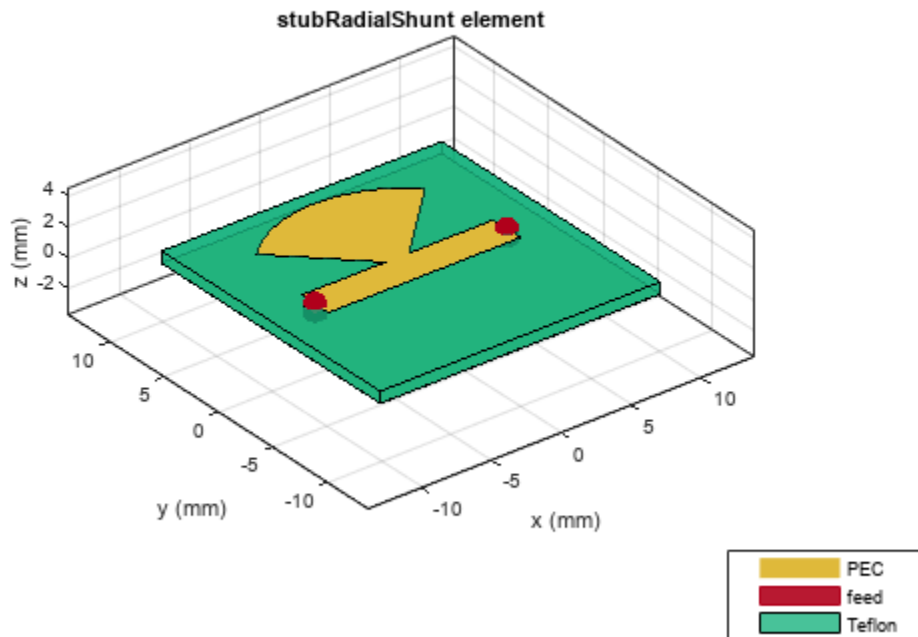
Create a default radial stub shunt.

```
stub = stubRadialShunt
```

```
stub =  
  stubRadialShunt with properties:  
  
    StubType: 'Single'  
   OuterRadius: 0.0085  
   InnerRadius: 0.0012  
      Angle: 90  
  PortLineLength: 0.0137  
  PortLineWidth: 0.0025  
      Height: 8.0000e-04  
GroundPlaneLength: 0.0200  
GroundPlaneWidth: 0.0200  
  Substrate: [1x1 dielectric]  
  Conductor: [1x1 metal]
```

View the radial stub shunt.

```
show(stub)
```



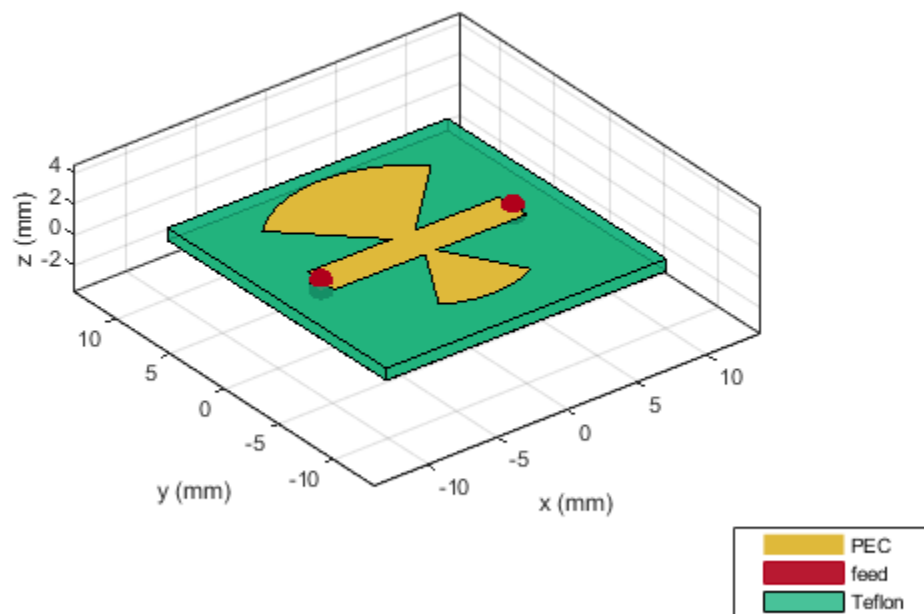
Create Double Shunt Radial Stub

Create shunt radial stub of type double.

```
stub = stubRadialShunt(StubType='double');  
stub.OuterRadius = [0.0085 0.0065];  
stub.InnerRadius = [0.0012 0.0008];  
stub.Angle       = [90 60];
```

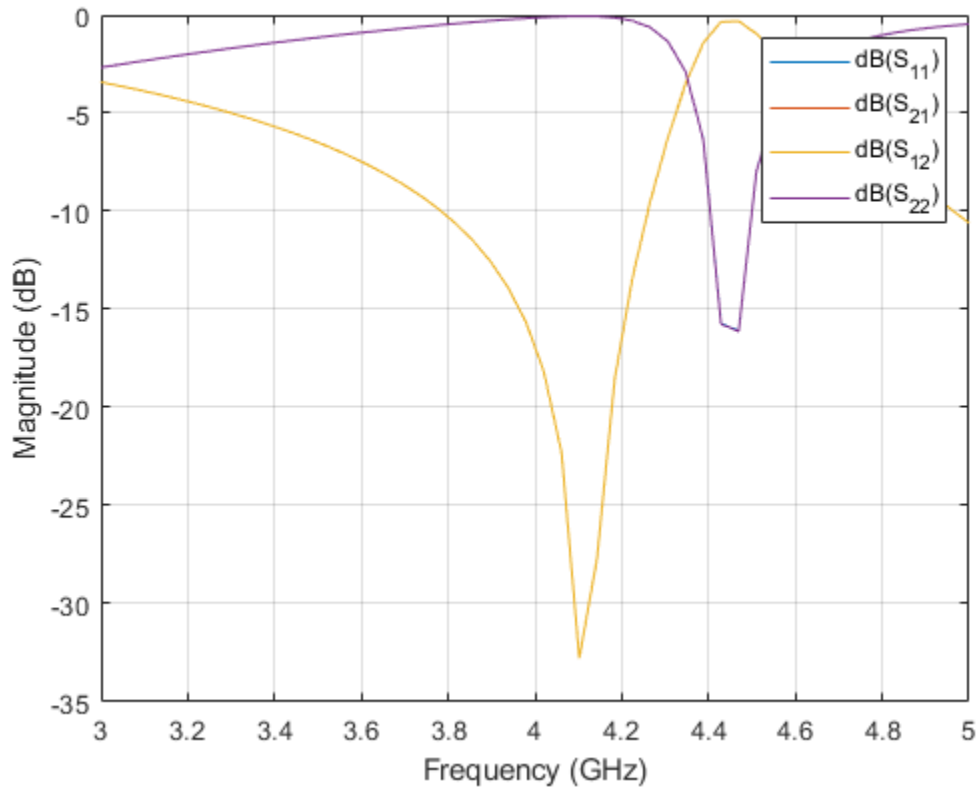
View shunt radial stub.

```
show(stub)
```



Plot s-parameters.

```
spar = sparameters(stub, linspace(3e9, 5e9, 50));  
rfplot(spar)
```



More About

Parametric Analysis Guidelines

- Increase the Angle to shift the resonance of the stub to a lower frequency.
- Increase the OuterRadius to shift the resonance of the stub to a lower frequency.
- Adjust the OuterRadius and increase the Angle to design a radial stub at a desired frequency, line length, low insertion loss, and wide bandwidth.

Version History

Introduced in R2021b

References

- [1] Wang , Zhebin, and Chan-Wang Park. "Novel Wideband GaN HEMT Power Amplifier Using Microstrip Radial Stub to Suppress Harmonics." In *2012 IEEE/MTT-S International Microwave Symposium Digest*, 1-3. Montreal, QC, Canada: IEEE, 2012. <https://doi.org/10.1109/MWSYM.2012.6259464>.
- [2] Singh, Prashant, and Tiwary Anjini. "Novel Compact Dual Bandstop Filter Using Radial Stub." *Microwave Review* 21 (September 1, 2015): 17-22.

See Also

microstripLine

traceSpiral

Create even-sided polygon trace in spiral form

Description

Use the `traceSpiral` object to create an even-sided polygon trace such as a square, hexagon, octagon, decagon, or a circle in a spiral form.

Creation

Syntax

```
trace = traceSpiral  
trace = traceSpiral(Name=Value)
```

Description

`trace = traceSpiral` creates a square spiral trace. The spiral trace is centered at the origin on the X-Y plane.

`trace = traceSpiral(Name=Value)` sets "Properties" on page 1-158 using one or more name-value arguments. For example, `traceSpiral(ReferencePoint=[1 1])` creates a spiral trace with the reference point [1 1]. Properties not specified retain their default values.

Properties

Name — Name of spiral trace

'myspiral' (default) | character vector | string scalar

Name of the spiral trace, specified as a character vector or a string scalar.

Example: `trace = traceSpiral(Name="spiraltrace1")`

Data Types: char | string

ReferencePoint — Point of reference of spiral trace

[0 0] (default) | two-element vector

Point of reference of the spiral trace in Cartesian coordinates, specified as a two-element vector. Use the reference point to modify the shape from its initial position.

Example: `trace = traceSpiral(ReferencePoint=[1 1])`

Data Types: double

InnerDiameter — Inner diameter of spiral trace

0.0040 (default) | positive scalar

Inner diameter of the spiral trace in meters, specified as a positive scalar. If the polygon is a square, the inner diameter is the distance between the innermost vertex and the midpoint of the opposite side

of the inner square. For all other shapes, the value is the distance between the innermost vertex and the opposite vertex of the inner turn.

Example: `trace = traceSpiral(InnerDiameter=0.0015)`

Data Types: double

TraceWidth — Width of spiral trace

0.0020 (default) | positive scalar

Width of the spiral trace in meters, specified as a positive scalar.

Example: `trace = traceSpiral(TraceWidth=0.0050)`

Data Types: double

Spacing — Distance between traces of spiral

5.0000e-04 (default) | positive scalar

Distance between the traces of the spiral in meters, specified as a positive scalar. For a square spiral trace, the spacing is the gap between the flat edges of adjacent turns. For all other shapes, the spacing is the gap between vertices of adjacent turns.

Example: `trace = traceSpiral(Spacing=6.0000e-04)`

Data Types: double

NumTurns — Number of turns in spiral

4 (default) | positive scalar

Number of turns in the spiral, specified as a positive scalar.

Example: `trace = traceSpiral(NumTurns=6)`

Data Types: double

NumSides — Number of sides in each turn

4 (default) | positive, even scalar

Number of sides in each turn of the spiral based on the polygon, specified as a positive, even scalar. The minimum number of sides is 4 and the maximum number is 10. If the number exceeds 10, then the shape is a circle.

Example: `trace = traceSpiral(NumSides=6)`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis

rotateY Rotate RF PCB shape about y-axis and angle
rotateZ Rotate RF PCB shape about z-axis
subtract Boolean subtraction operation on two RF PCB shapes
scale Change size of RF PCB shape by fixed amount
show Display PCB component structure or PCB shape
translate Move RF PCB shape to new location

Examples

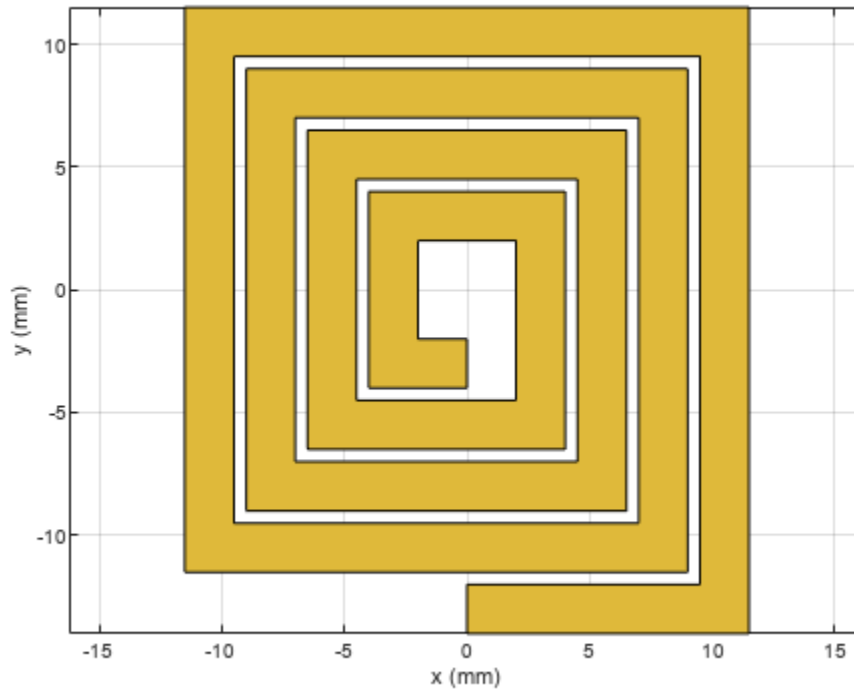
Create Default Spiral Trace

Create a spiral trace with default properties.

```
trace = traceSpiral  
  
trace =  
  traceSpiral with properties:  
      Name: 'myspiral'  
ReferencePoint: [0 0]  
  InnerDiameter: 0.0040  
  TraceWidth: 0.0020  
    Spacing: 5.0000e-04  
  NumTurns: 4  
  NumSides: 4
```

View the trace.

```
show(trace)
```



Version History

Introduced in R2021b

See Also

[traceLine](#) | [traceCross](#) | [traceTee](#) | [tracePoint](#) | [traceRectangular](#)

traceRectangular

Create rectangular trace

Description

Use the `traceRectangular` object to create a rectangular trace centered at the origin on the X-Y plane.

Creation

Syntax

```
trace = traceRectangular  
trace = traceRectangular(Name=Value)
```

Description

`trace = traceRectangular` creates a rectangular trace centered at the origin and on the X-Y plane.

`trace = traceRectangular(Name=Value)` sets “Properties” on page 1-162 using one or more name-value arguments. For example, `traceRectangular(Center=[1 1])` creates a rectangular trace centered at [1 1]. Properties not specified retain their default values.

Properties

Name — Name of rectangular trace

'mytraceRectangular' (default) | character vector | string scalar

Name of the rectangular trace, specified as a character vector or a string scalar.

Example: `trace = traceRectangular(Name="rectangleTrace1")`

Data Types: `char` | `string`

Center — Center of rectangular trace

[0 0] (default) | two-element vector

Center of the rectangular trace in Cartesian coordinates, specified as a two-element vector.

Example: `trace = traceRectangular(Center=[1 1])`

Data Types: `double`

Length — Length of rectangle

0.0200 (default) | positive scalar

Length of the rectangle in meters, specified as a positive scalar.

Example: `trace = traceRectangular(Length=0.0500)`

Data Types: double

Width — Width of rectangle

0.0050 (default) | positive scalar

Width of the rectangle in meters, specified as a positive scalar.

Example: `trace = traceRectangular(Width=0.015)`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples

Create Default Rectangular Trace

Create a rectangular trace with default properties.

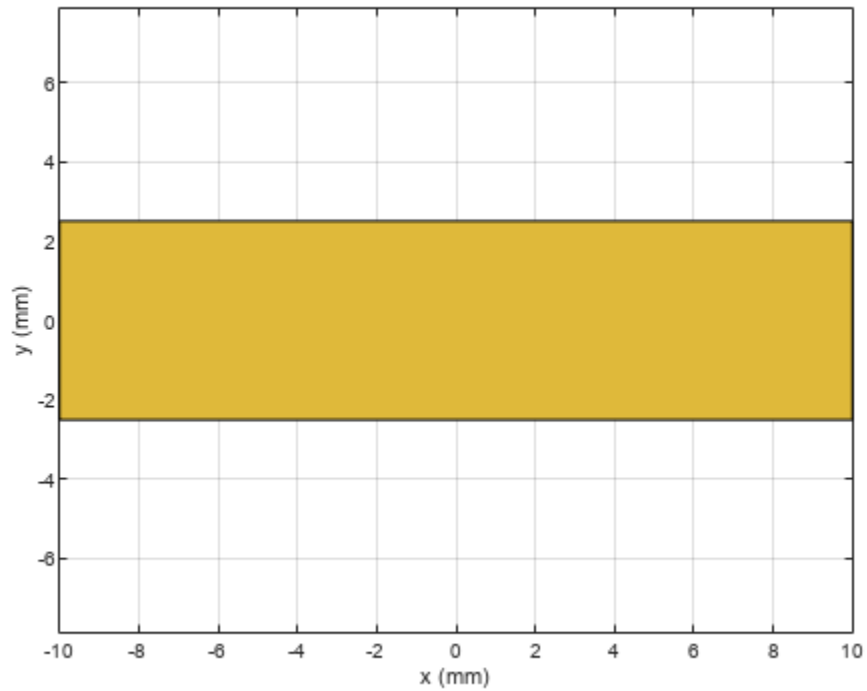
```
trace = traceRectangular

trace =
  traceRectangular with properties:

    Name: 'mytraceRectangular'
    Center: [0 0]
    Length: 0.0200
    Width: 0.0050
```

View the trace.

```
show(trace)
```



Mesh Rectangular Trace

Create a 2 cm-by-2 cm rectangular trace.

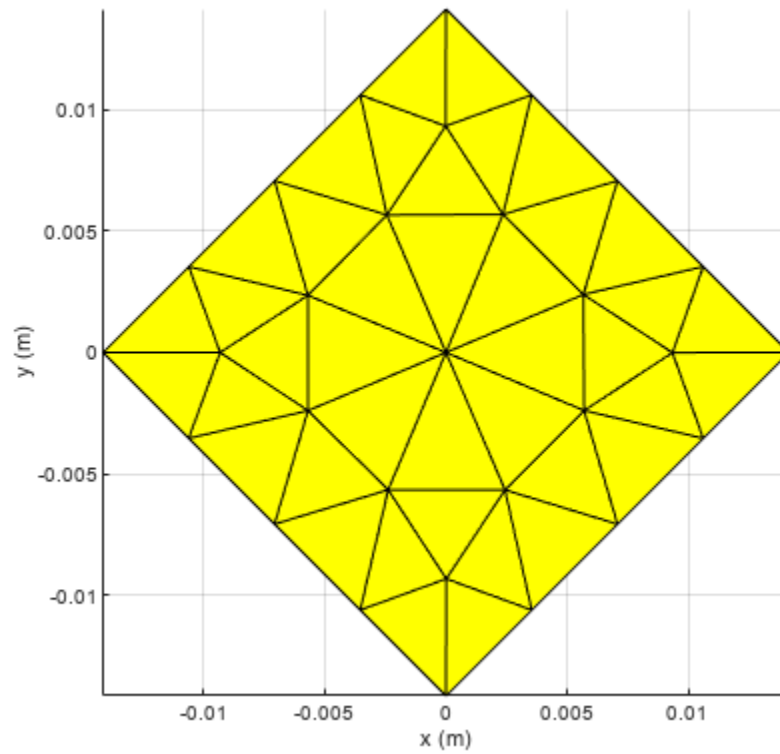
```
trace = traceRectangular(Length=0.02,Width=0.02);
```

Rotate the trace by 45 degrees about the z-axis.

```
trace = rotateZ(trace,45);
```

Mesh the trace at a maximum edge length of 5 mm.

```
mesh(trace,MaxEdgeLength=5e-3)
```

Version History

Introduced in R2021b

See Also

[traceLine](#) | [traceCross](#) | [traceTee](#) | [tracePoint](#) | [traceSpiral](#)

traceTee

Create tee trace

Description

Use the `traceTee` object to create a tee trace on the X-Y plane.

Note This shape object supports behavioral modeling. For more information, see “Behavioral Models”.

Creation

Syntax

```
trace = traceTee  
trace = traceTee(Name=Value)
```

Description

`trace = traceTee` creates a tee trace with default properties on the X-Y plane.

`trace = traceTee(Name=Value)` sets “Properties” on page 1-166 using one or more name-value arguments. For example, `traceTee(ReferencePoint=[1 1])` creates a tee trace with the reference point [1 1]. Properties not specified retain their default values.

Properties

Name — Name of tee trace

'mytraceTeeShape' (default) | character vector | string scalar

Name of the tee trace, specified as a character vector or a string scalar.

Example: `trace = traceTee(Name="traceTeeShape")`

Data Types: `char` | `string`

ReferencePoint — Reference point of tee trace

[0 0] (default) | two-element vector

Reference point of the tee trace in meters, specified as a two-element vector of nonnegative elements.

Example: `trace = traceTee(ReferencePoint=[1 1])`

Data Types: `double`

Length — Length of horizontal and vertical lines

[0.0200 0.0100] (default) | two-element vector

Length of the horizontal and vertical lines in meters, specified as a two-element vector of positive elements.

Example: `trace = traceTee(Length=[0.0300 0.0200])`

Data Types: double

Width — Width of horizontal and vertical lines

[0.0050 0.0050] (default) | two-element vector

Width of the horizontal and vertical lines in meters, specified as a two-element vector of positive elements.

Example: `trace = traceTee(Width=[0.0060 0.0060])`

Data Types: double

Offset — Offset along X-axis

0 (default) | nonnegative scalar

Offset along the X-axis in meters, specified as a nonnegative scalar.

Example: `trace = traceTee(Offset=0.0005)`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>translate</code>	Move RF PCB shape to new location
<code>scale</code>	Change size of RF PCB shape by fixed amount

Examples

Create Default Tee Trace

Create a tee trace with default properties.

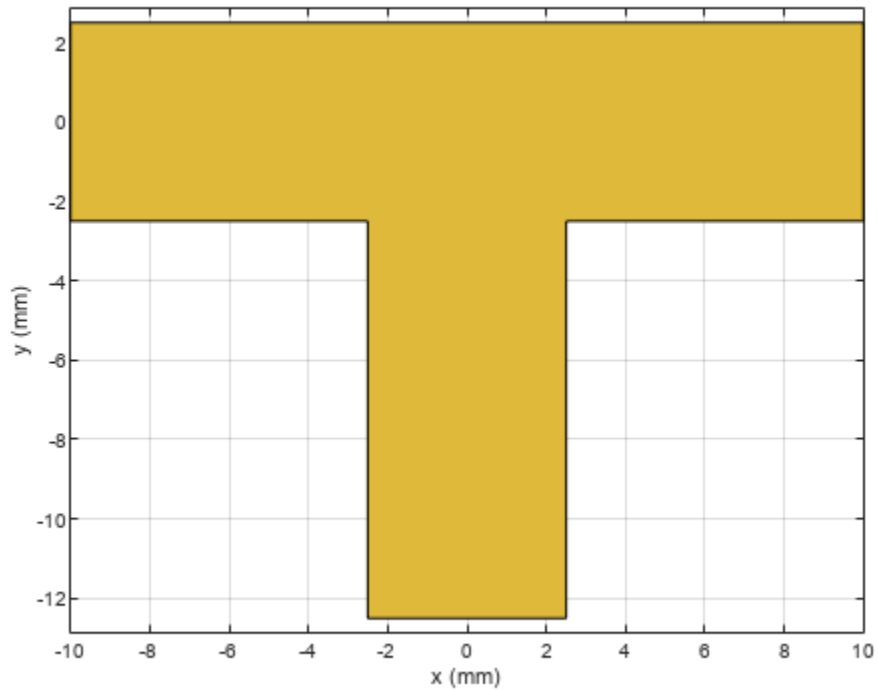
```
trace = traceTee
```

```
trace =
  traceTee with properties:
      Name: 'mytraceTeeShape'
  ReferencePoint: [0 0]
      Length: [0.0200 0.0100]
```

```
Width: [0.0050 0.0050]  
Offset: 0
```

View the trace.

```
show(trace)
```



Use Behavioral Model to Calculate S-Parameters of Microstrip T-Junction

Design a microstrip transmission line at 3 GHz for FR4 substrate.

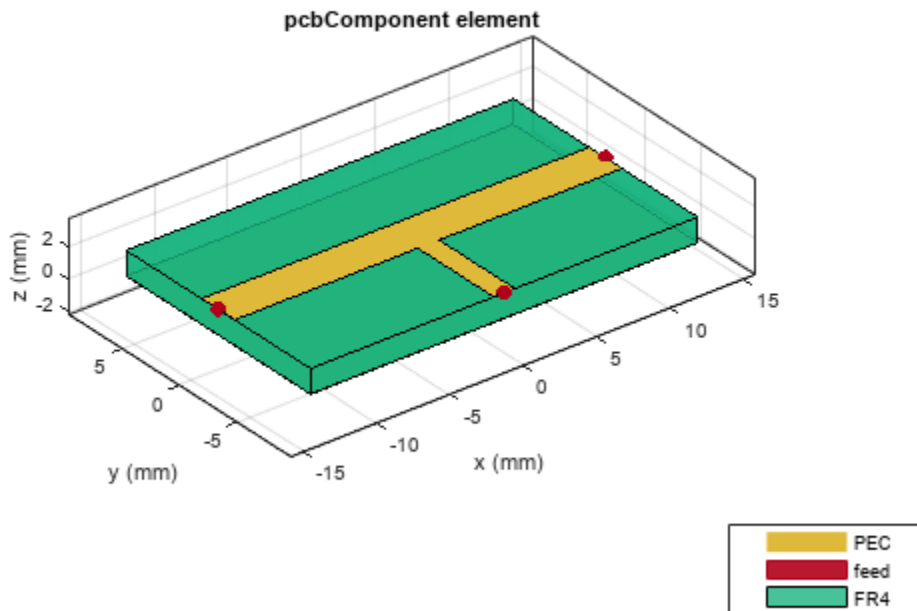
```
m = design(microstripLine('Substrate',dielectric('FR4')),3e9);
```

Create a microstrip T-junction.

```
layer2d = traceTee('Length',[m.Length m.Length/4],...  
"Width",[m.Width m.Width/2]);
```

Convert the T-junction trace to a 3-D component.

```
robj = pcbComponent(layer2d);  
robj.BoardThickness = m.Substrate.Thickness;  
robj.Layers{2} = m.Substrate;  
show(robj)
```



Define frequency points to calculate the s-parameters.

```
freq = (1:40)*100e6;
```

Calculate the s-parameters of the T-junction trace using the behavioral model.

```
Sckt = sparameters(robj, freq, 75, 'Behavioral', true);
```

Warning: Behavioral model is valid only when Z0 of main line is 50 ohms and for EpsilonR of 9.9.

Calculate the s-parameters of the T-junction trace using the electromagnetic solver.

```
Sem = sparameters(robj, freq, 75)
```

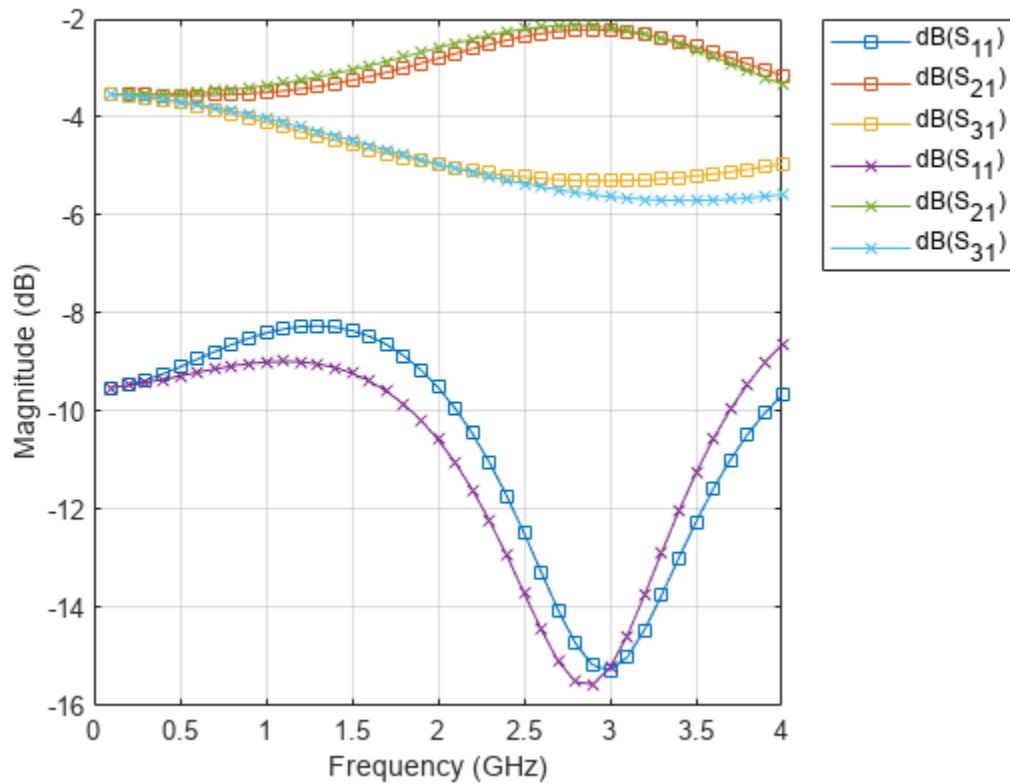
```
Sem =  
  sparameters: S-parameters object
```

```
  NumPorts: 3  
  Frequencies: [40x1 double]  
  Parameters: [3x3x40 double]  
  Impedance: 75
```

```
rfparam(obj,i,j) returns S-parameter Sij
```

Plot the s-parameter data using the `rfplot` function.

```
rfplot(Sckt,1:3,1,'db','-s')  
hold on  
rfplot(Sem,1:3,1,'db','-x')
```



References:

- 1 Ramesh Garg & I. J. Bahl (1978) Microstrip discontinuities, International Journal of Electronics, 45:1, 81-87, DOI: [10.1080/00207217808900883](https://doi.org/10.1080/00207217808900883)
- 2 Wadell, Brian C. *Transmission Line Design Handbook*. The Artech House Microwave Library. Boston: Artech House, 1991.

Version History

Introduced in R2021b

See Also

[traceLine](#) | [traceCross](#) | [traceRectangular](#) | [tracePoint](#) | [traceSpiral](#)

traceCross

Create cross-shaped trace

Description

Use the `traceCross` object to create a cross-shaped trace on the X-Y plane.

Note This shape object supports behavioral modeling. For more information, see “Behavioral Models”.

Creation

Syntax

```
trace = traceCross
trace = traceCross(Name=Value)
```

Description

`trace = traceCross` creates a cross-shaped trace with default properties on the X-Y plane.

`trace = traceCross(Name=Value)` sets “Properties” on page 1-171 using one or more name-value arguments. For example, `traceCross(ReferencePoint=[1 1])` creates a cross-shaped trace at the reference point [1 1]. Properties not specified retain their default values.

Properties

Name — Name of cross-shaped trace

'mytraceCross' (default) | character vector | string scalar

Name of the cross-shaped trace, specified as a character vector or a string scalar.

Example: `trace = traceCross(Name="traceCrossShape")`

Data Types: `char` | `string`

ReferencePoint — Reference point for cross-shaped trace

[0 0] (default) | two-element vector

Reference point for the cross-shaped trace in Cartesian coordinates, specified as a two-element vector.

Example: `trace = traceCross(ReferencePoint=[1 1])`

Data Types: `double`

Length — Length of cross-shaped trace

[0.0100 0.0100] (default) | two-element vector

Length of the cross-shaped trace in meters, specified as a two-element vector of positive elements.

Example: `trace = traceCross(Length=[0.0800 0.0400])`

Data Types: double

Width — Width of cross-shaped trace

`[0.0020 0.0020]` (default) | two-element vector

Width of the cross-shaped trace in meters, specified as a two-element vector of positive elements.

Example: `trace = traceCross(Width=[0.005 0.005])`

Data Types: double

Offset — Offset along X and Y direction

`[0 0]` (default) | two-element vector

Offset along the X and Y direction in meters, specified as a two-element vector.

Example: `trace = traceCross(Offset=[1 1])`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples

Create Default Cross Trace

Create a cross-shaped trace with default properties.

```
trace = traceCross
```

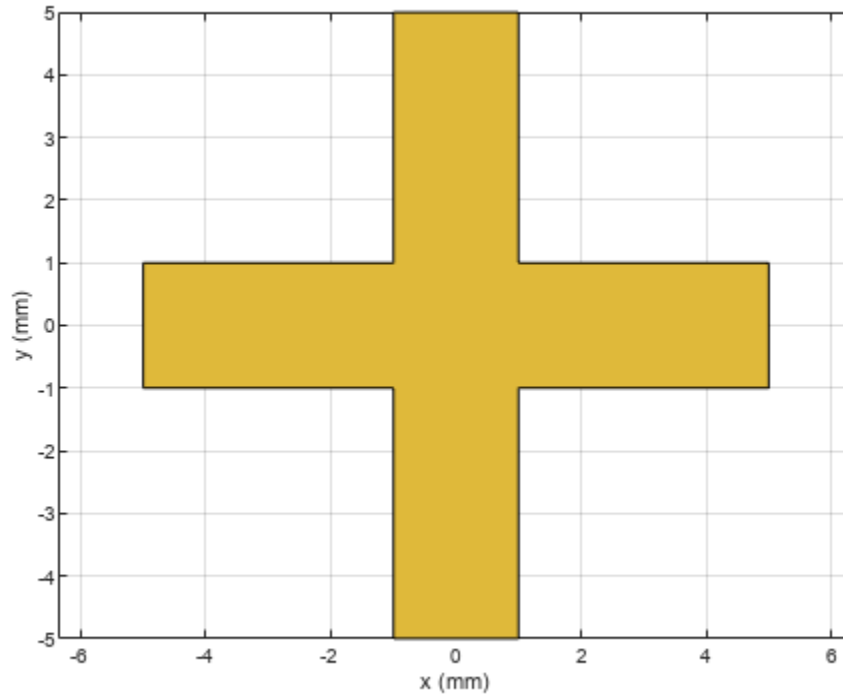
```
trace =  
  traceCross with properties:  
      Name: 'mytraceCross'  
  ReferencePoint: [0 0]  
      Length: [0.0100 0.0100]
```



```
Width: [0.0020 0.0020]
Offset: [0 0]
```

View the trace.

```
show(trace)
```



Use Behavioral Model to Calculate S-Parameters of Microstrip Cross

Design a microstrip transmission line at 3 GHz for FR4 substrate.

```
d = dielectric('FR4');
d.LossTangent = 0;
m = design(microstripLine('Substrate',d),3e9,'Z0',75,...
    'LineLength',0.5);
```

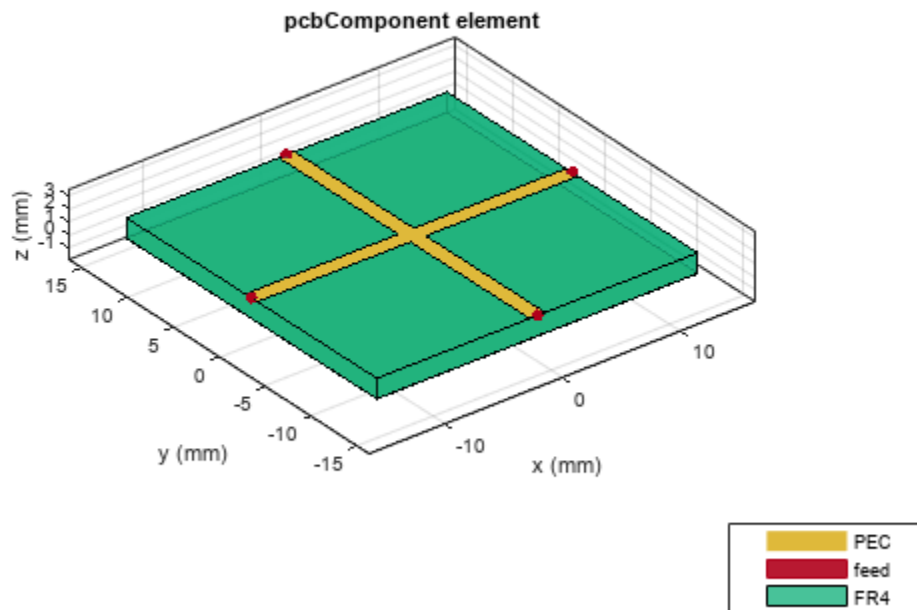
Create a microstrip cross.

```
layer2d = traceCross('Length',[m.Length m.Length], ...
    'Width',[m.Width m.Width]);
```

Convert the cross trace to a PCB component.

```
robj = pcbComponent(layer2d);
robj.BoardThickness = m.Substrate.Thickness;
```

```
robject.Layers{2} = m.Substrate;  
show(robject)
```



Define frequency points to calculate the s-parameters.

```
freq = (1:3:40)*100e6;
```

Calculate the s-parameters of the cross trace using the behavioral model.

```
Sckt = sparameters(robject, freq, 'Behavioral', true);
```

Warning: Behavioral model is valid only when EpsilonR is 9.9.

Calculate the s-parameters of the cross trace using the electromagnetic solver.

```
Sem = sparameters(robject, freq);
```

References:

- 1 Ramesh Garg & I. J. Bahl (1978) Microstrip discontinuities, *International Journal of Electronics*, 45:1, 81-87, DOI: [10.1080/00207217808900883](https://doi.org/10.1080/00207217808900883)
- 2 Wadell, Brian C. *Transmission Line Design Handbook*. The Artech House Microwave Library. Boston: Artech House, 1991.

Version History

Introduced in R2021b

See Also

[traceTee](#) | [traceRectangular](#) | [traceSpiral](#) | [tracePoint](#) | [traceLine](#)

delta

Create delta shape

Description

Use `delta` object to create a delta shape on the X-Y plane.

Creation

Syntax

```
deltashape = delta  
deltashape = delta(Name=Value)
```

Description

`deltashape = delta` creates a delta shape on the X-Y plane.

`deltashape = delta(Name=Value)` sets “Properties” on page 1-176 using one or more name-value arguments. For example, `delta(ReferencePoint=[1 1])` creates a delta shape with the reference point at [1 1]. Properties not specified retain their default values.

Properties

Name — Name of delta shape

'mydelta' (default) | character vector | string scalar

Name of the delta shape, specified as a character vector or string scalar.

Example: `deltashape = delta(Name='deltaShape')`

Data Types: char

ReferencePoint — Reference point of delta shape

[0 0] (default) | two-element vector

Reference point of delta shape in Cartesian coordinates, specified as a two-element vector of nonnegative elements. Use the reference point to modify the shape relative to its initial position.

Example: `deltashape = delta(ReferencePoint=[1 1])`

Data Types: double

OuterRadius — Outer radius of delta

0.0016 (default) | positive scalar

Outer radius of the delta, specified as a positive scalar in meters.

Example: `shape = delta(OuterRadius=0.0024)`

Data Types: double

InnerRadius — Inner radius of delta

0 (default) | nonnegative scalar

Inner radius of the delta, specified as a nonnegative scalar in meters. This value truncates the delta from the tip.

Example: `shape = delta(InnerRadius=0.3)`

Data Types: double

Angle — Angel of delta

90 (default) | positive scalar

Angel of the delta shape, specified as a positive scalar in degrees. The value of the angle must be greater than 0 degrees and lesser than 180 degrees.

Example: `shape = delta(Angle=50)`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples**Create Default Delta Shape**

Create a delta shape with default properties.

```
deltashape = delta
```

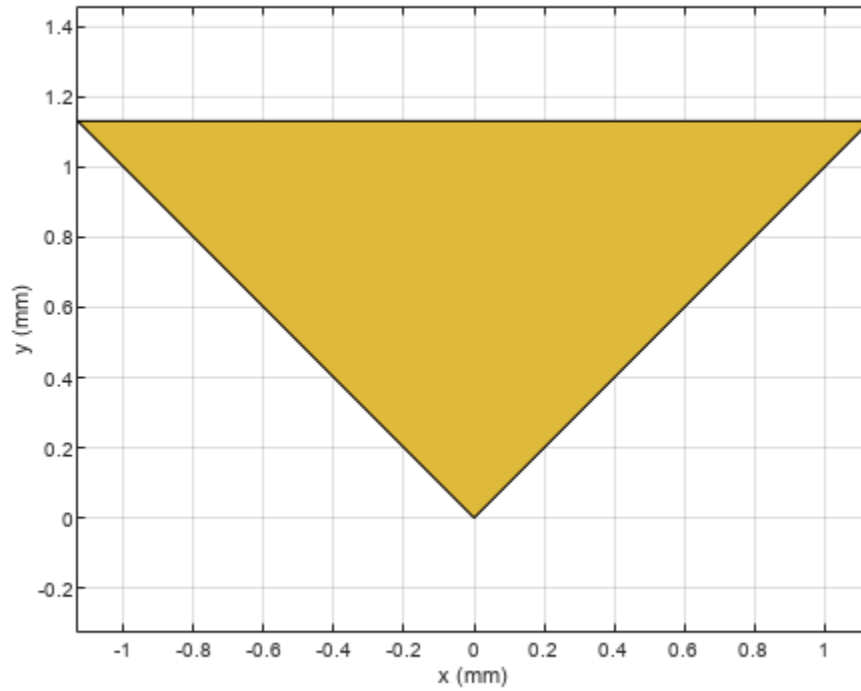
```
deltashape =
  delta with properties:
```

```

      Name: 'mydelta'
ReferencePoint: [0 0]
      OuterRadius: 0.0016
      InnerRadius: 0
      Angle: 90
```

View the shape.

```
show(deltashape)
```



Version History

Introduced in R2021b

See Also

`delta` | `radial`

radial

Create radial shape

Description

Use the `radial` object to create a radial shape on the X-Y plane.

Creation

Syntax

```
radialshape = radial  
radialshape = radial(Name=Value)
```

Description

`radialshape = radial` creates a radial shape on the X-Y plane.

`radialshape = radial(Name=Value)` sets “Properties” on page 1-179 using one or more name-value arguments. For example, `radial(ReferencePoint=[1 1])` creates a radial shape with the reference point at [1 1]. Properties not specified retain their default values.

Properties

Name — Name of radial shape

'myradial' (default) | character vector | string scalar

Name of the radial shape, specified as a character vector or string scalar.

Example: `radialshape = radial(Name='radialShape')`

Data Types: `char` | `string`

ReferencePoint — Reference point of radial shape

[0 0] (default) | two-element vector

Reference point of radial shape, specified as a two-element vector of nonnegative elements in Cartesian coordinates. Use the reference point to modify the shape relative to its initial position.

Example: `radialshape = radial(ReferencePoint=[1 1])`

Data Types: `double`

OuterRadius — Outer radius of radial

0.0016 (default) | positive scalar

Outer radius of the radial shape, specified as a positive scalar in meters.

Example: `radialshape = radial(OuterRadius=0.0024)`

Data Types: `double`

InnerRadius — Inner radius of radial

0 (default) | positive scalar

Inner radius of the radial shape, specified as a positive scalar in meters. This value truncates the radial from the tip.

Example: `radialshape = radial(InnerRadius=0.4)`

Data Types: double

Angle — Angel of radial

90 (default) | positive scalar

Angel of the radial shape, specified as a positive scalar in degrees. The value of the angle must be greater than 0 degrees and lesser than 180 degrees.

Example: `radialshape = radial(Angle=50)`

Data Types: double

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>scale</code>	Change size of RF PCB shape by fixed amount
<code>show</code>	Display PCB component structure or PCB shape
<code>translate</code>	Move RF PCB shape to new location

Examples**Create Default Radial Shape**

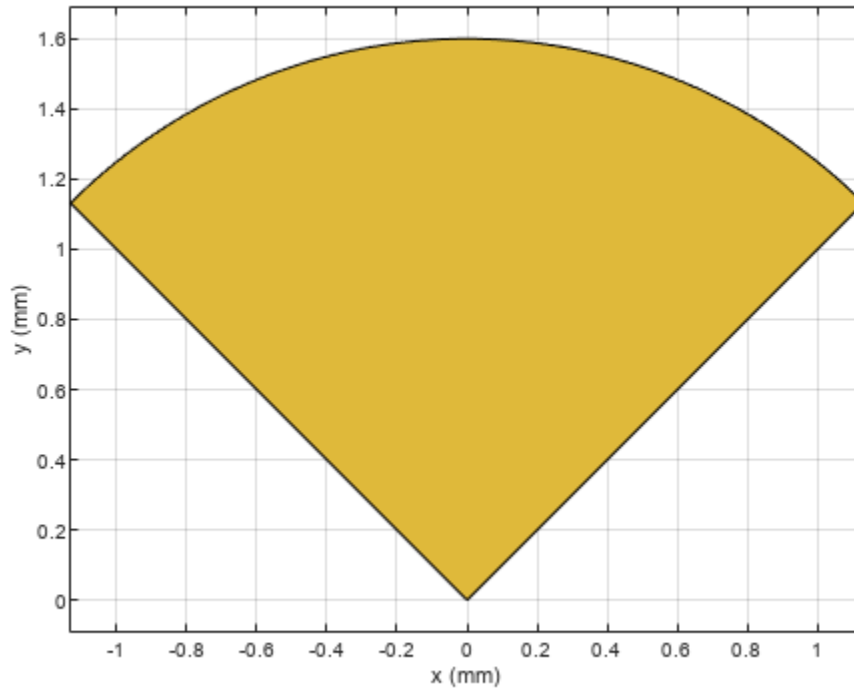
Create a radial shape with default properties.

```
radialshape = radial
```

```
radialshape =  
  radial with properties:  
  
      Name: 'myradial'  
ReferencePoint: [0 0]  
  OuterRadius: 0.0016  
  InnerRadius: 0  
      Angle: 90
```


View the shape.

```
show(radialshape)
```



Version History

Introduced in R2021b

See Also

delta

PCBConnectors

RF connector at RF PCB feedpoint

Description

Use PCBConnectors object to specify RF connectors used for RF printed circuit board (PCB) feed points. The result is generally a set of modifications to the PCB design files. The changes to the PCB include new copper landing pads and traces, and changes to solder mask, silk screen, and solder paste files.

Creation

Syntax

```
c = PCBConnectors.connectorType
```

Description

`c = PCBConnectors.connectorType` creates Gerber files based on the type of connector to use at the PCB feedpoint specified in `connectorType`.

Input Arguments

connectorType — Type of connector from PCB connector package

character vector

Type of connector from PCB connector package, specified as one of the following:

- Coax Connectors - Coax RG11, RG174, RG58, and RG59 connectors directly soldered to PCB pads.
- IPX Connectors - LightHorse IPX SMT jack or plug surface mount RF connector.
- MMCX Connectors - MMCX Cinch or Samtec surface mount RF connectors.
- SMA Connectors - Generic 5-pad SMA surface mount RF connectors, with four corner rectangular pads, one round center pin. Cinch and Multicomp SMA RF connectors.
- SMAEdge Connectors- Generic SMA edge-launch surface mount RF connector. Cinch and Samtec SMA edge-launch RF connectors.
- SMB Connectors - Johnson/Emerson and Pasternack SMB surface mount RF connectors.
- SMC Connectors - Pasternack SMC and SMC edge-launch surface mount RF connectors.
- Coaxial Cable Connectors - Semi-rigid 0.020 inch, 0.034 inch, 0.047 inch, and 0.118 inch coaxial cable soldered to PCB pads.

For list of connectors, see “PCB Connectors List” on page 1-189.

Example: `c = PCBConnectors.Semi_020` creates Gerber files configured to use semi-rigid 0.020 inch coaxial cables.

Properties

Common Properties for All Connectors

Type — Type of connector

character vector

This property is read-only.

Type of connector, specified as a character vector.

Example: 'Coax_RG11'

Data Types: char | string

Mfg — Name of component manufacturer

character vector

This property is read-only.

Name of component manufacturer, specified as a character vector.

Example: 'Belden'

Data Types: char | string

Part — Manufacturer part number

character vector | string

This property is read-only.

Manufacturer part number, specified as a character vector or string.

Example: 'RG11'

Data Types: char | string

Annotation — Text added to PCB to identify component

character vector

This property is read-only.

Text added to PCB to identify component, specified as a character vector.

Example: 'RG59U'

Data Types: char | string

Impedance — Connector impedance

50 | positive scalar

This property is read-only.

Connector impedance, specified as a positive scalar in ohms.

Example: `c = PCBConnectors.MMCX_Cinch; c.Impedance = 70;`

Data Types: double

Datasheet — URL for component specifications

character vector

This property is read-only.

URL for component specifications, specified as a character vector. Data sheets are typically PDF files.

Data Types: char | string

Purchase — URL for purchasing connector

character vector

This property is read-only.

URL for purchasing connector, specified as a character vector.

Data Types: char | string

Common Properties for All Coax Connectors**PinDiameter — Circular pad diameter**

positive scalar

Circular pad diameter connecting the signal wire of the coax to the feedpoint, specified as a positive scalar in meters. The pin diameter is greater than the diameter of the signal wire.

Example: `c = PCBConnectors.Coax_RG59; c.PinDiameter = 1.0000e-03;`

Data Types: double

DielectricDiameter — Dielectric diameter

positive scalar

Dielectric diameter (white material around signal wire), specified as a positive scalar in meters. Dielectric diameter specifies the size of the non-conductive isolation ring on the PCB between the signal wire and the ground plane.

Example: `c = PCBConnectors.Coax_RG59; c.DielectricDiameter = 0.0073;`

Data Types: double

ShieldDiameter — Ground ring diameter

positive scalar

Ground ring diameters used to solder coax shield, specified as a positive scalar in meters.

Example: `c = PCBConnectors.Coax_RG59; c.ShieldDiameter = 0.0085;`

Data Types: double

AddThermals — Thermal relief

1 | 0

Thermal relief around coaxial shield connection, specified as 0 or 1. Thermal relief reduces the heat needed to solder the coax shield to the ground.

Example: `c = PCBConnectors.Coax_RG59; c.AddThermals = 0;`

Data Types: logical

ThermalsDiameter — Arc-shaped gaps outer diameter

positive scalar

Arc-shaped gaps outer diameter in the ground plane, specified as a positive scalar in meters.

Example: `c = PCBConnectors.Coax_RG59; c.ThermalsDiameter = 0.0100;`

Data Types: double

ThermalsBridgeWidth — Width of four conductive bridges

positive scalar

Width of four conductive bridges created across thermal gap, specified as a positive scalar in meters. The bridges are established during electrical grounding.

Example: `c = PCBConnectors.Coax_RG59; c.ThermalBridgeWidth = 0.0015;`

Data Types: double

Common Properties for All 5-Pad Symmetric Surface Mount Connectors**TotalSize — Total length of each side of rectangular connector footprint**

two-element vector

Total length of each side of rectangular connector footprint, specified as a two-element vector with each element unit in meters.

Example: `c = PCBConnectors.SMA_Multicomp; c.TotalSize = [0.0063 0.0063];`

Data Types: double

GroundPadSize — Length of each side of ground pad

two-element vector

Length of each side of ground pad, specified as a two-element vector with each element unit in meters. The pads are located in each of the four corners of the connector footprint.

Example: `c = PCBConnectors.SMA_Multicomp; c.GroundPadSize = [0.0016 0.0016];`

Data Types: double

SignalPadDiameter — Circular pad diameter

positive scalar

Circular pad diameter connecting the signal pin of the coax connector, specified as a positive scalar in meters. The pad is at the center of the connector footprint.

Example: `c = PCBConnectors.SMA_Multicomp; c.SignalPadDiameter = 0.0012;`

Data Types: double

PinHoleDiameter — Via pin diameter

positive scalar

Via pin diameter, specified as a positive scalar in meters.

Example: `c = PCBConnectors.SMA_Multicomp; c.ViaPinDiameter = 0.0012;`

Data Types: double

IsolationRing — Diameter of isolation ring that removes semicircle of copper from inner corner of ground pads

scalar

Diameter of isolation ring that removes semicircle of copper from inner corner of ground pads, specified as a scalar in meters.

```
Example: c = PCBConnectors.SMA_Multicomp; c.IsolationRing = 0.0012;
```

Data Types: double

VerticalGroundStrips — Vertical ground strips between upper and lower ground pads

scalar

Vertical ground strips between upper and lower ground pads, specified as a scalar.

```
Example: c = PCBConnectors.SMA_Multicomp; c.VerticalGroundStrips = 1;
```

Data Types: double

Common Properties for All Edge-Launch Surface Mount Connectors**GroundPadSize — Ground pad size**

two-element vector

Ground pad size, specified as a two-element vector with each element unit in meters.

```
Example: c = PCBConnectors.SMAEdge; c.GroundPadSize = [0.0014 0.0042];
```

Data Types: double

GroundSeparation — Space between ground pads

positive scalar

Space between ground pads on the ground side of the board, specified as a positive scalar in meters.

```
Example: c = PCBConnectors.SMAEdge; c.GroundSeparation = 0.0043;
```

Data Types: double

GroundPadIsolation — Width of copper removed around top layer ground pads

positive scalar

Width of copper removed around top layer ground pads, specified as a positive scalar in meters. This property isolates the ground pads from any signal traces or structures.

```
Example: c = PCBConnectors.SMAEdge; c.GroundPadIsolation = 2.5000e-04;
```

Data Types: double

SignalPadSize — Signal pad size

two-element vector

Signal pad size, specified as a two-element vector with each element unit in meters.

```
Example: c = PCBConnectors.SMAEdge; c.SignalPadSize = [0.0013 0.0036];
```

Data Types: double

SignalGap — Gap between PCB edge and start of signal pad copper

positive scalar

Gap between PCB edge and start of signal pad copper, specified as a positive scalar in meters.

Example: `c = PCBConnectors.SMAEdge; c.SignalGap = 1.0000e-04;`

Data Types: double

SignalLineWidth — Width of signal trace

positive scalar

Width of signal trace extending from the signal pad to the feedpoint location, specified as a positive scalar in meters.

Example: `c = PCBConnectors.SMAEdge; c.SignalLineWidth = 8.0000e-04;`

Data Types: double

EdgeLocation — PCB side that receives edge connector

'north' | 'south' | 'east' | 'west'

PCB side that receives edge connector, specified as 'north', 'south', 'east', 'west'.

Example: `c = PCBConnectors.SMAEdge; c.EdgeLocation = 'south';`

Data Types: char

EdgeBoardProfile — Extend PCB to add connector beyond design area

0 | 1

Extend PCB to add connector beyond design area, specified as 0 or 1

Example: `c = PCBConnectors.SMAEdge; c.EdgeBoardProfile = 1;`

Data Types: logical

FillGroundSide — Fill connector region on ground side of board with copper

0 | 1

Fill connector region on ground side of the board with copper, specified as 0 or 1

Example: `c = PCBConnectors.SMAEdge; c.FillGroundSide = 1;`

Data Types: logical

Common Properties for All Staggered Surface Mount Connectors

GroundPadSize — Ground pad size

two-element vector

Ground pad size, specified as a two-element vector with each element unit in meters.

Example: `c = PCBConnectors.IPX_Plug_Lighthouse; c.GroundPadSize = [0.0010 0.0022];`

Data Types: double

GroundPadXSeparation — Distance between pair of ground pads along X-axis

positive scalar

Distance between pair of ground pads along X-axis, specified as a positive scalar in meters.

Example: `c = PCBConnectors.IPX_Plug_Lighthouse; c.GroundPadXSeparation = 0.0019;`

Data Types: double

GroundPadYOffset — Y-offset from signal pad to signal pad center line

positive scalar

Y-offset from signal pad to signal pad center line, specified as a positive scalar in meters.

Example: `c = PCBConnectors.IPX_Plug_Lighthouse; c.GroundPadYOffset = 0.0015;`

Data Types: double

SignalPadSize — Signal pad size

2-element vector

Signal pad size, specified as a 2-element vector with each element unit in meters.

Example: `c = PCBConnectors.IPX_Plug_Lighthouse; c.SignalPadSize = [1.0000e-03 1.0000e-03];`

Data Types: double

SignalMinYSeparation — Minimum separation from ground at bottom or top for signal pad

positive scalar

Minimum separation from ground at bottom or top for signal pad, specified as a positive scalar in meters.

Example: `c = PCBConnectors.IPX_Plug_Lighthouse; c.SignalMinYSeparation = 1.0000e-03;`

Data Types: double

Examples

Authoring Custom RF Connector

This example shows how to define custom RF connector class.

```
classdef SMA_Jack_Cinch < PCBConnectors.BaseSMT5PadSymmetric
    % Cinch SMA surface mount RF connector.

    properties (Constant) % Abstract
        Type          = 'SMA'
        Mfg            = 'Cinch'
        Part           = '142-0701-631'
        Annotation     = 'SMA'
        Impedance      = 50
        Datasheet      = 'http://www.farnell.com/datasheets/1720451.pdf?_ga=2.164811836.2075200750.142-0701-631'
        Purchase       = 'http://www.newark.com/johnson/142-0701-631/rf-coaxial-sma-jack-straight-50'
    end

    methods
        function RFC = SMA_Jack_Cinch
            RFC.TotalSize          = [0.5 0.5]*25.4e-3;
            RFC.GroundPadSize      = [0.102 0.102]*25.4e-3;
            RFC.SignalPadDiameter = 0.1*25.4e-3;
            RFC.PinHoleDiameter   = 1.27e-3;
        end
    end
end
```



```

        RFC.IsolationRing      = 0.22*25.4e-3;
        RFC.VerticalGroundStrips = false;
    end
end
end

```

Generate Gerber Format Files for Default PCB Component

Create a default PCB component.

```
p = pcbComponent;
```

Use 2 Cinch SMA connectors and the Mayhew Labs PCB viewer.

```

W = PCBServices.MayhewWriter;
C1 = PCBConnectors.SMA_Cinch;
C2 = PCBConnectors.SMA_Cinch;

```

Generate the Gerber-format files.

```
[A,g] = gerberWrite(p,W,{C1,C2})
```

A =

PCBWriter with properties:

```

        Design: [1x1 struct]
        Writer: [1x1 PCBServices.MayhewWriter]
        Connector: {1x2 cell}
        UseDefaultConnector: 0
        ComponentBoundaryLineWidth: 8
        ComponentNameFontSize: []
        DesignInfoFontSize: []
        Font: 'Arial'
        PCBMargin: 5.0000e-04
        Soldermask: 'both'
        Solderpaste: 1

```

See info for details

g =

'C:\TEMP\Bdoc22b_2054784_6060\ibB18F8B\7\tpc19835a0\rfpcb-ex06685827\untitled'

More About

PCB Connectors List

PCB Connectors	Descriptions
PCBConnectors.Coax_RG11	RG11 coaxial cable direct soldered to PCB pads.
PCBConnectors.Coax_RG58	RG58 coaxial cable direct soldered to PCB pads.
PCBConnectors.Coax_RG59	RG59 coaxial cable direct soldered to PCB pads.
PCBConnectors.Coax_RG174	RG174 coaxial cable direct soldered to PCB pads.

PCB Connectors	Descriptions
PCBConnectors.SMA	Generic 5-pad SMA surface mount RF connector, with four corner rectangular ground pads, one round.
PCBConnectors.SMAEdge	Generic SMA edge-launch surface mount RF connector.
PCBConnectors.SMACinch	Cinch SMA surface mount RF connector
PCBConnectors.SMAEdge_Cinch	Cinch SMA edge-launch surface mount RF connector
PCBConnectors.SMAEdge_Samtec	Samtec SMA edge-launch surface mount RF connector
PCBConnectors.SMAEdge_Amphenol	Amphenol SMA edge-launch surface mount RF connector
PCBConnectors.SMAEdge_Linx	Linx SMA edge-launch surface mount RF connector
PCBConnectors.SMA_Multicomp	Multicomp SMA surface mount RF connector
PCBConnectors.SMB_Johnson	Johnson/Emerson SMB surface mount RF connector
PCBConnectors.SMB_Pasternack	Pasternack SMB surface mount RF connector
PCBConnectors.SMC_Pasternack	Pasternack SMC surface mount RF connector
PCBConnectors.SMCEdge_Pasternack	Pasternack SMC edge-launch surface mount RF connector
PCBConnectors.MMCX_Cinch	Cinch MMCX surface mount RF connector
PCBConnectors.MMCX_Samtec	Samtec MMCX surface mount RF connector
PCBConnectors.IPX_Jack_LightHorse	LightHorse IPX SMT jack surface mount RF connector
PCBConnectors.IPX_Plug_LightHorse	LightHorse IPX SMT plug surface mount RF connector
PCBConnectors.UFL_Hirose	Hirose u.fl surface mount RF connector
PCBConnectors.Semi_020	Pasternack semi-rigid 0.020" coaxial cable soldered to PCB pads
PCBConnectors.Semi_034	Pasternack semi-rigid 0.020" coaxial cable soldered to PCB pads
PCBConnectors.Semi_047	Pasternack semi-rigid 0.047" coaxial cable soldered to PCB pads
PCBConnectors.Semi_118	Pasternack semi-rigid 0.118" coaxial cable soldered to PCB pads

Version History

Introduced in R2021b

See Also

PCBWriter | PCBServices | gerberWrite

PCBReader

Import and update Gerber files

Description

Use the `PCBReader` object to create a printed circuit board (PCB) reader to import Gerber files and to facilitate the creation of a PCB model. A Gerber file is a set of manufacturing files used to describe a PCB. A Gerber file uses an ASCII vector format to describe 2-D binary images.

Creation

You can create a `PCBReader` object using the following methods:

- `gerberRead` — Create a `PCBReader` object with the specified Gerber and drill files.
- The `PCBReader` function described here.

Syntax

```
B = PCBReader(S)  
B = PCBReader(Name=Value)
```

Description

`B = PCBReader(S)` creates a `PCBReader` object that imports multilayer PCB design files described in `S`.

Note

- To translate the center of an imported symmetrical or asymmetrical polygon to $[0,0]$, please use one of the following MATLAB® functions: `boundingbox` and `centroid`. See examples, “Translate Center of Imported Symmetrical Polygon to $[0,0]$ ” on page 1-194 and “Translate Center of Imported Asymmetrical Polygon to $[0,0]$ ” on page 1-196.
 - The `PCBReader` object reads RS-274X Gerber files. It does not support RS-274D Gerber files.
-

`B = PCBReader(Name=Value)` sets “Properties” on page 1-193 using name-value arguments. `Name` is the property name and `Value` is the corresponding value. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Properties not specified retain their default values. For example, `B = PCBReader('StackUp',S,'Drillfile','ant.txt')` imports the layer and drill files into the `PCBReader`.

Input Arguments

S — PCB stackup definition

`stackUp` object

PCB stackup definition, specified as a `stackUp` object.

Example: `S = stackUp; B = PCBReader(S)`

Example: `B = PCBReader('StackUp',S)`

Properties

StackUp — PCB stackup definition

`stackUp` object

PCB stackup definition, specified as a `stackUp` object.

Example: `S = stackUp; B.StackUp = S;`

Example: `B = PCBReader('StackUp',S)`

DrillFile — Name of Excellon drill file

[] (default) | character vector | string scalar

Name of Excellon drill file, specified as a character vector or string scalar. You can specify either a DRL or a TXT file.

Example: `B.DrillFile = 'ant.drl'`

NumPointsOnCurves — Discretization points on curved segments

50 (default) | positive scalar

Discretization points on curved segments, specified as a positive scalar.

Example: `B.NumPointsOnCurves = 80`

Examples

Import Gerber Files Using Stackup Definition

Create a PCB stack up definition object using default properties.

```
S = stackUp;
```

Set the thickness of the dielectric Air in layer 1 to 0.1 mm.

```
S.Layer1.Thickness = 0.1e-3;
```

Import a top layer Gerber file to layer 2.

```
S.Layer2 = 'interdigital_Capacitor.gtl';
```

Create a `PCBReader` object using the `stackUp` object, `S`.

```
p = PCBReader('StackUp',S);
```

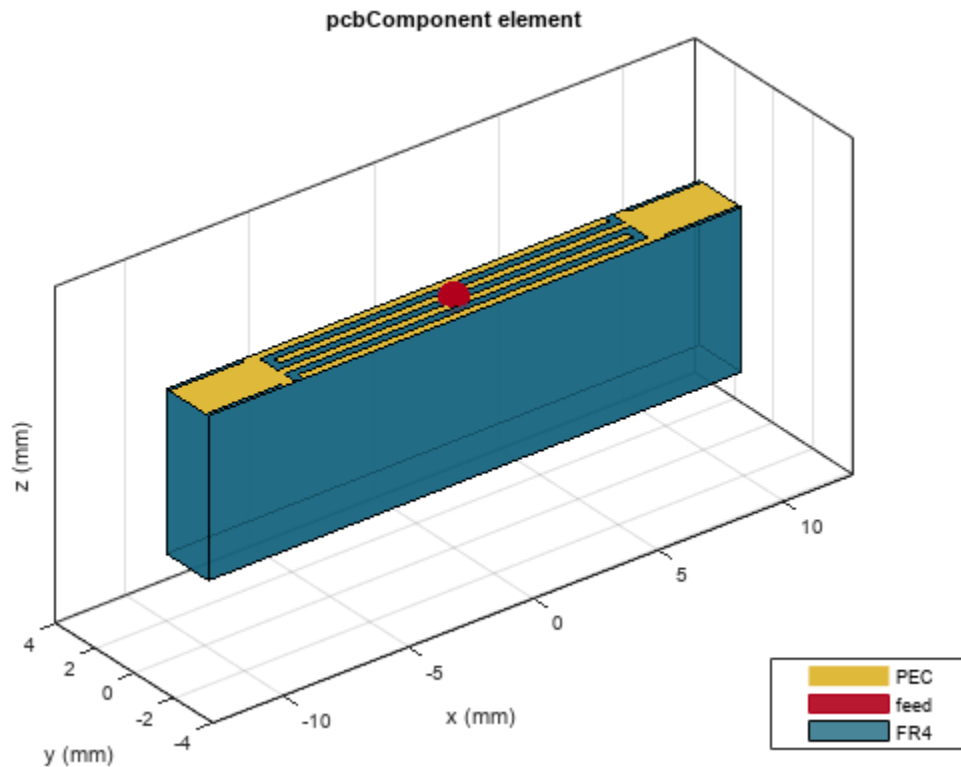
To update the Gerber file, convert the `PCBReader` object to a `pcbComponent` object.

```
pcbcapacitor = pcbComponent(p);
pcbcapacitor.FeedDiameter = 0.001
```

```
pcbcapacitor =  
  pcbComponent with properties:  
  
      Name: 'interdigital_Capacitor'  
      Revision: 'v1.0'  
      BoardShape: [1x1 antenna.Rectangle]  
      BoardThickness: 0.0061  
      Layers: {1x4 cell}  
      FeedLocations: [0 0 2]  
      FeedDiameter: 1.0000e-03  
      ViaLocations: []  
      ViaDiameter: []  
      FeedViaModel: 'square'  
      Conductor: [1x1 metal]  
      Tilt: 0  
      TiltAxis: [0 0 1]  
      Load: [1x1 lumpedElement]
```

View the PCB component in the Gerber file.

```
show(pcbcapacitor)
```



Translate Center of Imported Symmetrical Polygon to [0,0]

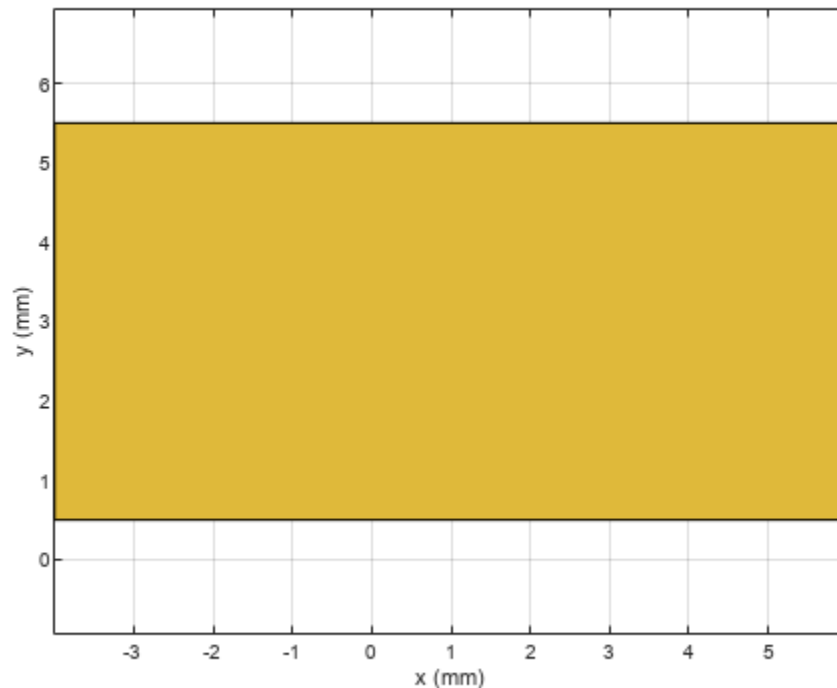
This example will show how to translate the symmetrical polygon imported from the Gerber file to the respective co-ordinates.

Create a PCB stackup and import rectangular patch on it.

```
S = stackUp;
S.Layer2 = 'PatchRectangular.gtl';
S.Layer3 = dielectric('Teflon');
```

Use a PCB Reader to read the polygon shape from the stackup.

```
p1 = PCBReader ('StackUp',S);
figure; show(p1.shapes);
```

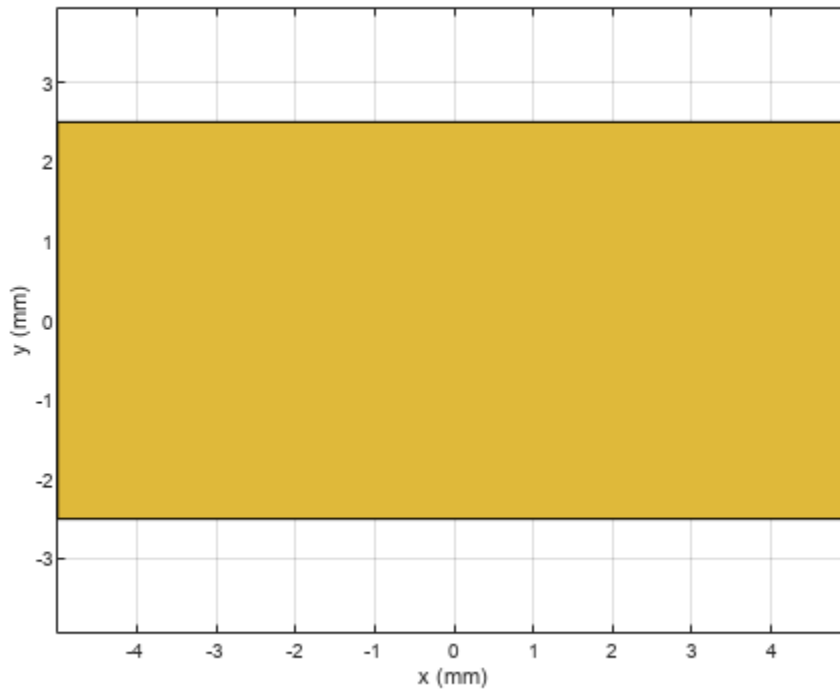


Translate the shape with center (0,0) using the `centroid` function from MATLAB.

```
s = p1.shapes
```

```
s =
  Polygon with properties:
    Name: 'mypolygon'
    Vertices: [4x3 double]
```

```
polygon = s;  
[x,y] = centroid(polygon);  
translate(polygon,[-x, -y, 0]);
```



Translate Center of Imported Asymmetrical Polygon to [0,0]

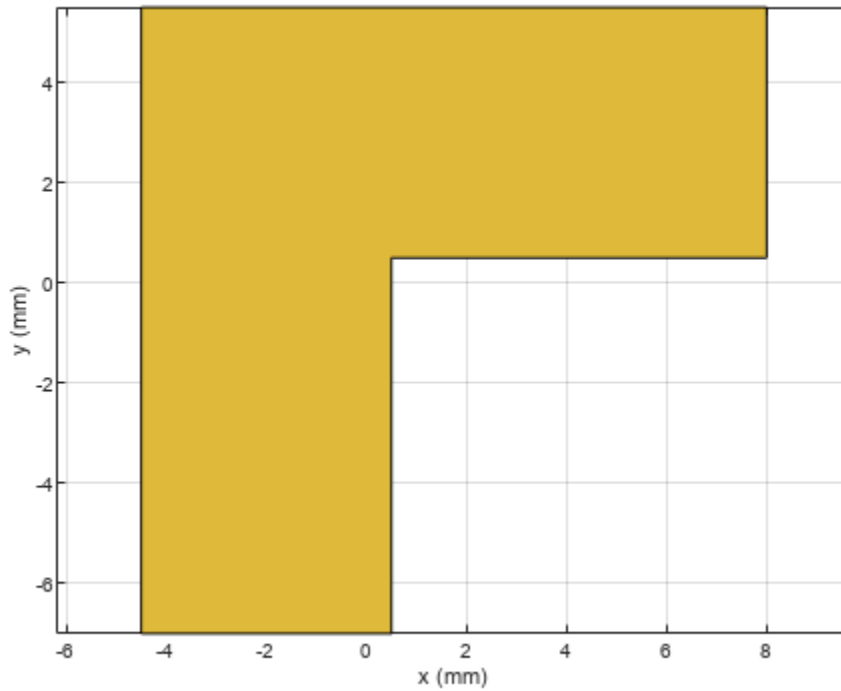
This example shows how to translate the asymmetrical polygon imported from the Gerber file to the respective co-ordinates.

Create a PCB stackup and import rectangular patch on it.

```
S = stackUp;  
S.Layer2 = 'RightAngleBend.gtl';  
S.Layer3 = dielectric('Teflon');
```

Use a PCB Reader to read the polygon shape from the stackup.

```
p1 = PCBReader ('StackUp',S);  
figure; show(p1.shapes);
```

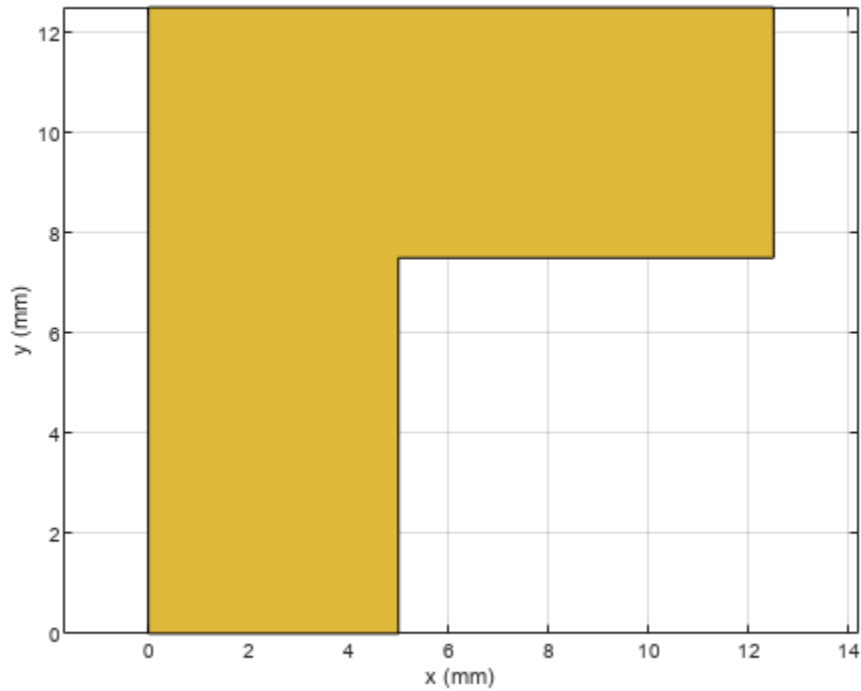
Translate the shape's bottom left corner to (0,0). Use the boundingbox function from MATLAB to convert the shape to polyshape and find the upper and lower bounds of the shape.

```
s = pl.shapes
```

```
s =  
Polygon with properties:
```

```
    Name: 'mypolygon'  
  Vertices: [6x3 double]
```

```
ver = s.Vertices(:,1:2);  
polygon = polyshape(ver);  
[xlim, ylim] = boundingbox(polygon);  
translate(s,[-xlim(1), -ylim(1), 0]);
```



Version History

Introduced in R2021b

See Also

[PCBWriter](#) | [PCBServices](#) | [PCBConnectors](#) | [stackUp](#) | [gerberRead](#)

PCBServices

Customize PCB file generation for PCB manufacturing service

Description

Use the `PCBServices` object to customize printed circuit board (PCB) file generation for a PCB manufacturing service.

Creation

Syntax

```
w = PCBServices.serviceType
```

Description

`w = PCBServices.serviceType` creates a Gerber file based on the type of service specified in `serviceType`.

Input Arguments

serviceType — Type of service from PCB services package

character vector

Type of service from PCB services package, specified as one of the following:

- `AdvancedCircuitsWriter` - Configure Gerber file generation for Advanced Circuits manufacturing.
- `CircuitPeopleWriter` - Configure Gerber file generation for CircuitPeople online viewer.
- `DirtyPCBsWriter` - Configure Gerber file generation for Dirty PCBs manufacturing.
- `EuroCircuitsWriter` - Configure Gerber file generation for EuroCircuits online viewer.
- `GerberLookWriter` - Configure Gerber file generation for GerbLook online viewer.
- `GerberViewerWriter` - Configure Gerber file generation for GerberViewer online viewer.
- `MayhewWriter` - Configure Gerber file generation for Mayhew Labs online 3-D viewer.
- `OSHParkWriter` - Configure Gerber file generation for OSH Park PCB manufacturing.
- `PCBWayWriter` - Configure Gerber file generation for PCBWay PCB manufacturing.
- `ParagonWriter` - Configure Gerber file generation for Paragon Robotics online viewer.
- `SeedWriter` - Configure Gerber file generation for Seed Fusion PCB manufacturing.
- `SunstoneWriter` - Configure Gerber file generation for Sunstone PCB manufacturing.
- `ZofzWriter` - Configure Gerber file generation for Zofz 3-D viewer.

Example: `w = PCBServices.SunstoneWriter` creates Gerber files configured to use Sunstone PCB manufacturing service.

Output Arguments

w — PCB manufacturing service

object

PCB manufacturing service, returned as an object.

Properties

BoardProfileFile — File type for board profile

'legend' | 'profile'

File type for board profile, specified as 'legend' or 'profile'.

Example: `w = PCBServices.SunstoneWriter; w.BoardProfileFile = 'profile'.`

Data Types: char | string

BoardProfileLineWidth — Width of line

1 | positive scalar

Width of line, specified as a positive scalar in mils.

PCB manufacturers vary on board profile. The most common line width is zero of a fraction width in the chosen unit, for example, 0.1 mil.

Example: `w = PCBServices.SunstoneWriter; w.BoardProfileLineWidth = 0.1`

Data Types: double

CoordPrecision — Precision of X and Y coordinates written to file

[2 6] | 1-by-2 vector

Precision of X and Y coordinates written to file, specified as a 1-by-2 vector $[I F]$, where,

- I - Number of digits in the integer part, $0 \leq I \leq 6$.
- F - Number of digits in the fractional part, $4 \leq F \leq 6$.

Example: `w = PCBServices.SunstoneWriter; w.CoordPrecision = [1 3]`

Data Types: double

CoordUnits — Units of X and Y coordinate

'in' | 'mm'

Units of X and Y coordinates, specified as inches or millimeters.

Example: `w = PCBServices.SunstoneWriter; w.CoordUnits = 'mm'`

Data Types: char | string

CreateArchiveFile — Creates single archive file with all Gerber files

1 (default) | 0

Creates single archive file with all Gerber files, specified as 1 or 0.

Example: `w = PCBServices.SunstoneWriter; w.CreateArchiveFile = 0`

Data Types: logical

DefaultViaDiameter — Via drill diameter

3.0000e-04 | positive scalar

Via drill diameter, specified as a positive scalar in meters. PCB manufacturers also call it minimum drilling hole diameter.

Example: `w = PCBServices.SunstoneWriter; w.DefaultViaDiameter = 0.1`

Data Types: double

DrawArcsUsingLines — Force arcs to be drawn using lines

0 | 1

Force arcs to be drawn using lines, specified as 1 or 0.

Example: `w = PCBServices.SunstoneWriter; w.DrawArcsUsingLines = 0`

Data Types: logical

ExtensionLevel — Feature content for Gerber file format

1 (default) | 2

Feature content for Gerber file format, specified as:

- 1 - Extension 1 is the most compatible setting for downstream PCB manufacturing tools.
- 2 - Extension 2 adds file attributes `%TF.<attr>*%` to the header and footer of Gerber files.

Example: `w = PCBServices.SunstoneWriter; w.ExtensionLevel = 2`

Data Types: double

Filename — Name of all files containing Gerber design

'untitled' (default) | character vector

Name of all files containing Gerber design, specified as a character vector.

Example: `w = PCBServices.SunstoneWriter; w.Filename = 'pcb_design'`.

Data Types: char | string

Files — Define stack of PCB files

character vector

Define stack of PCB files, specified as a character vector. This definition includes:

- Multiple files describing one PCB.
- A "file" as a memory object containing buffers that describe or hold the file content before the file is written.
- Cell vector of `Gerber.FileFunction` objects, one per file.

Data Types: cell | char | string

IncludeRootFolderInZip — Include top-level folder in zip archive

1 | 0

Include top-level folder in zip archive, specified as 1 or 0.

Example: `w = PCBServices.SunstoneWriter; w.IncludeRootFolderInZip = 0`

Data Types: logical

PostWriteFcn — Function to invoke after a successful write operation

function handle (default)

Function to invoke after a successful write operation, specified as a function handle. In this case, it is the `sendTo` function. This property makes sure that the location of the Gerber files and the website of the manufacturing service is open after a successful write function.

Example: `w = PCBServices.SunstoneWriter; w.PostWriteFcn = @(obj)sendTo(obj)`

Data Types: function_handle

SameExtensionForGerberFiles — Use .gbr to be file extension for all Gerber files

0 | 1

Use `.gbr` to be file extension for all Gerber files, specified as 0 or 1.

Example: `w = PCBServices.SunstoneWriter; w.SameExtensionForGerberFiles = 1`

Data Types: logical

UseExcellon — Generate Excellon drill files

1 | 0

Generate Excellon drill files, specified as 0 or 1.

Example: `w = PCBServices.SunstoneWriter; w.UseExcellon = 1`, generates Gerber format drill files with 'x2' extension.

Data Types: logical

Examples

Generate Gerber Format Files for Default PCB Component

Create a default PCB component.

```
p = pcbComponent;
```

Use 2 Cinch SMA connectors and the Mayhew Labs PCB viewer.

```
W = PCBServices.MayhewWriter;  
C1 = PCBConnectors.SMA_Cinch;  
C2 = PCBConnectors.SMA_Cinch;
```

Generate the Gerber-format files.

```
[A,g] = gerberWrite(p,W,{C1,C2})
```

A =

PCBWriter with properties:

```
Design: [1x1 struct]  
Writer: [1x1 PCBServices.MayhewWriter]  
Connector: {1x2 cell}  
UseDefaultConnector: 0  
ComponentBoundaryLineWidth: 8
```

```
ComponentNameFontSize: []
DesignInfoFontSize: []
    Font: 'Arial'
    PCBMargin: 5.0000e-04
    Soldermask: 'both'
    Solderpaste: 1
```

See info for details

```
g =
'C:\TEMP\Bdoc22b_2054784_6060\ibB18F8B\7\tpc19835a0\rfpcb-ex06685827\untitled'
```

Version History

Introduced in R2021b

See Also

PCBWriter | PCBConnectors | gerberWrite

PCBWriter

Create PCB board definitions from 2-D PCB designs

Description

Use the `PCBWriter` object to create a printed circuit board (PCB) design files based on multilayer 2-D PCB design. A set of manufacturing files known as Gerber files describes a PCB. A Gerber file uses an ASCII vector format for 2-D binary images.

Creation

Syntax

```
b = PCBWriter(pcbcomponentObject)
b = PCBWriter(pcbcomponentObject, rfConnector)
b = PCBWriter(pcbcomponentObject, writer)
b = PCBWriter(pcbcomponentObject, rfConnector, writer)
```

Description

`b = PCBWriter(pcbcomponentObject)` creates a `PCBWriter` object that generates Gerber-format PCB design files based on a 2-D PCB design geometry using PCB stack.

`b = PCBWriter(pcbcomponentObject, rfConnector)` creates a customized PCB file using specified `rfConnector` type.

`b = PCBWriter(pcbcomponentObject, writer)` creates a customized PCB file using a specified PCB service, `writer`.

`b = PCBWriter(pcbcomponentObject, rfConnector, writer)` creates customised PCB file using specified PCB service and PCB connector type.

Input Arguments

pcbcomponentObject — Single feed PCB

`pcbComponent` object

Single feed PCB, specified as a `pcbComponent` object.

Example: `p1 = pcbComponent` creates a PCB component object, `p1`. `a = PCBWriter(p1)` uses `p1` to create a `PCBWriter` object `a`.

writer — PCB service to view PCB design

`PCBServices` object

PCB service to view PCB design, specified as a `PCBServices` object.

Example: `s = PCBServices.MayhewWriter`; `a = PCBWriter(p1, s)` uses Mayhew Labs PCB service to view the PCB design.

rfConnector — RF connector type

PCBConnectors object

RF connector type for PCB feedpoint, specified as a PCBConnectors object.

Example: `c = PCBConnectors.SMA_Cinch; a = PCBWriter(p1,c)` uses SMA_Cinch RF connector at feedpoint.

Properties**UseDefaultConnector — Use default connector**

1 (default) | 0

Use default connector, specified as 0 or 1.

Example: `a.UseDefaultConnector = 1`, where `a` is a PCBWriter object.

Data Types: logical

ComponentBoundaryLineWidth — Line widths drawn around components on silk screens

8 (default) | positive scalar

Line widths drawn around components on silk screens, specified as a positive scalar in mils.

Example: `a.ComponentBoundaryLineWidth = 10`, where `a` is a PCBWriter object.

Data Types: double

ComponentNameFontSize — Font size to label components on silk screen

positive scalar

Font size to label components on silk screen, specified as a positive scalar in points.

Example: `a.ComponentNameFontSize = 12`, where `a` is a PCBWriter object.

Data Types: double

DesignInfoFontSize — Font size for design informzation added outside board profile

positive scalar

Design information text font size added outside board profile, specified as a positive scalar.

Example: `a.DesignInfoFontSize = 12`, where `a` is a PCBWriter object.

Data Types: double

Font — Font used for component name and design info

'Arial' (default) | character vector

Font used for component name and design info, specified as a character vector.

Example: `a.Font = 'TimesNewRoman'`, where `a` is a PCBWriter object.

Data Types: char | string

PCBMargin — Copper free margin around board

0.5e-3 (default) | positive scalar

Copper free margin around board, specified as a positive scalar in meters.

Example: `a.PCBMargin = 0.7e-3`, where `a` is a `PCBWriter` object.

Data Types: `double`

SolderMask — Add solder mask to top and bottom of PCB

`'both'` (default) | `'top'` | `'bottom'` | `'none'`

Add solder mask to top and bottom of PCB, specified as `'both'`, `'top'`, `'bottom'` or `'none'`.

Example: `a.SolderMask = 'top'`, where `a` is a `PCBWriter` object.

Data Types: `char` | `string`

SolderPaste — Generate solder paste files

`1` (default) | `0`

Generate solder paste files as a part of PCB stack, specified as `1` or `0`.

Example: `a.SolderPaste = 0`, where `a` is a `PCBWriter` object.

Data Types: `logical`

Object Functions

`gerberWrite` Generate Gerber files

Examples

Create PCB Component Design File Using Mayhew Manufacturing Services

Create a coplanar waveguide.

```
cpw = coplanarWaveguide
```

```
cpw =  
  coplanarWaveguide with properties:  
    Length: 0.0231  
    Width: 0.0039  
    Spacing: 2.0000e-04  
    ViaSpacing: [0.0011 0.0070]  
    ViaDiameter: 5.0000e-04  
    Height: 0.0016  
    GroundPlaneWidth: 0.0300  
    Substrate: [1x1 dielectric]  
    Conductor: [1x1 metal]
```

Use this waveguide to create a `pcbComponent` object.

```
p = pcbComponent(cpw);  
p.Name = 'Coplanar Waveguide'
```

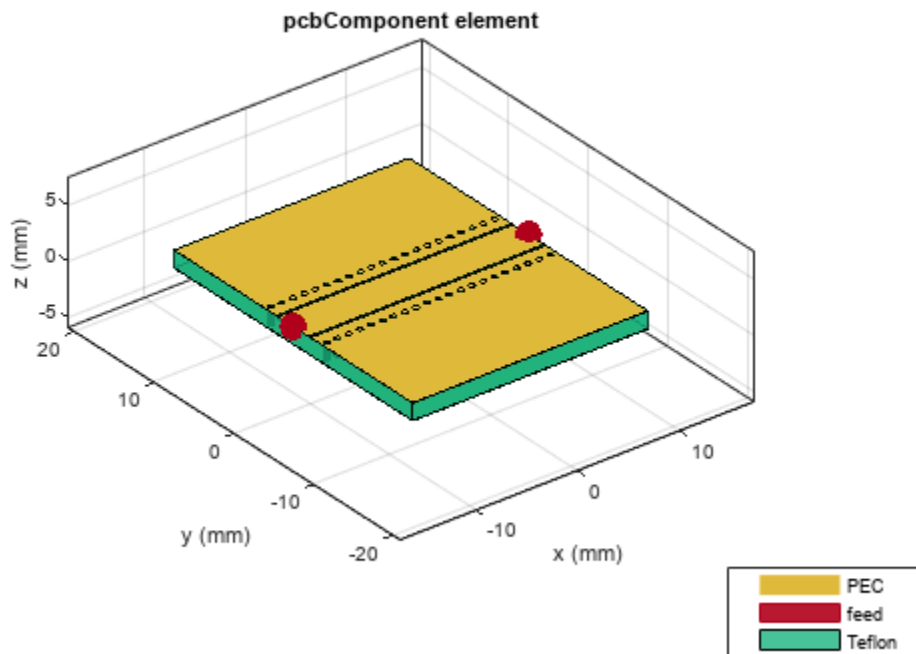
```
p =  
  pcbComponent with properties:  
    Name: 'Coplanar Waveguide'  
    Revision: 'v1.0'
```

```

BoardShape: [1x1 antenna.Rectangle]
BoardThickness: 0.0016
Layers: {1x3 cell}
FeedLocations: [2x4 double]
FeedDiameter: 0.0019
ViaLocations: [42x4 double]
ViaDiameter: 5.0000e-04
FeedViaModel: 'strip'
Conductor: [1x1 metal]
Tilt: 0
TiltAxis: [0 0 1]
Load: [1x1 lumpedElement]

```

```
show(p)
```



Use an SMA_Cinch as an RF connector and Mayhew Writer as a 3-D viewer.

```
c = PCBConnectors.SMA_Cinch
```

```
c =
SMA_Cinch with properties:
```

```

Type: 'SMA'
Mfg: 'Cinch'
Part: '142-0711-202'
Annotation: 'SMA'
Impedance: 50

```

```
Datasheet: 'https://belfuse.com/resources/Johnson/drawings/dr-142-0711-202.pdf'  
Purchase: 'https://www.digikey.com/product-detail/en/cinch-connectivity-solutions/  
TotalSize: [0.0071 0.0071]  
GroundPadSize: [0.0024 0.0024]  
SignalPadDiameter: 0.0017  
PinHoleDiameter: 0.0013  
IsolationRing: 0.0041  
VerticalGroundStrips: 1
```

Cinch 142-0711-202 (Example Purchase)

```
s = PCBServices.MayhewWriter
```

```
s =
```

```
MayhewWriter with properties:
```

```
BoardProfileFile: 'legend'  
BoardProfileLineWidth: 1  
CoordPrecision: [2 6]  
CoordUnits: 'in'  
CreateArchiveFile: 0  
DefaultViaDiam: 3.0000e-04  
DrawArcsUsingLines: 1  
ExtensionLevel: 1  
Filename: 'untitled'  
Files: {}  
IncludeRootFolderInZip: 0  
PostWriteFcn: @(obj)sendTo(obj)  
SameExtensionForGerberFiles: 0  
UseExcellon: 1
```

Create a PCB component design file.

```
PW = PCBWriter(p,s,c)
```

```
PW =
```

```
PCBWriter with properties:
```

```
Design: [1x1 struct]  
Writer: [1x1 PCBServices.MayhewWriter]  
Connector: {1x2 cell}  
UseDefaultConnector: 0  
ComponentBoundaryLineWidth: 8  
ComponentNameFontSize: []  
DesignInfoFontSize: []  
Font: 'Arial'  
PCBMargin: 5.0000e-04  
Soldermask: 'both'  
Solderpaste: 1
```

See info for details

Version History

Introduced in R2021b

See Also

PCBServices | PCBConnectors

stackUp

Create PCB stackup definition

Description

Use the `stackUp` object to create a printed circuit board (PCB) stackup definition to import Gerber files. A Gerber file is a set of manufacturing files used to describe a PCB. A Gerber file uses an ASCII vector format for 2-D binary images.

Creation

Syntax

```
s = stackUp
```

Description

`s = stackUp` creates a default PCB stackup object with five layers. Specify Gerber files as inputs to the second and fourth layers. Specify dielectric material objects as inputs to layers one, three, and five.

Properties

NumLayers — Number of layers in stackup

5 (default) | positive scalar

This property is read-only.

Number of layers in the stackup, returned as a positive scalar.

Layer1 — First layer in stackup

'Air' (default) | dielectric object

First layer in the stackup definition object, specified as a dielectric object.

```
Example: s = stackUp; d = dielectric('R04725JXR'); s.Layer1 = d;
```

Layer2 — Second layer in stackup

character vector | string scalar

Second layer in the stackup definition object, specified as a character vector or string. The file should be saved as a GTL, GBL, or GBR file.

```
Example: s = stackUp; s.Layer2 = 'antenna_design_file.gtl';
```

Note The Gerber file must be imported to the MATLAB workspace before setting this property.

Layer3 – Third layer in stackup

'FR4' (default) | dielectric object

Third layer in the stackup definition object, specified as a dielectric object.

Example: `s = stackUp; d = dielectric('R04725JXR'); s.Layer3 = d;`

Layer4 – Fourth layer in stackup

character vector | string scalar

Fourth layer in the stackup definition object, specified as a character vector or string. The file should be saved as a GTL, GBL, or GBR file.

Example: `s = stackUp; s.Layer4 = 'antenna_design_file.gbl';`

Note The Gerber file must be imported to the MATLAB workspace before setting this property.

Layer5 – Fifth layer in stackup

'Air' (default) | dielectric object

Fifth layer in the stackup definition object, specified as a dielectric object.

Example: `s = stackUp; d = dielectric('R04725JXR'); s.Layer5 = d;`

Examples**Import Gerber Files Using Stackup Definition**

Create a PCB stack up definition object using default properties.

```
S = stackUp;
```

Set the thickness of the dielectric Air in layer 1 to 0.1 mm.

```
S.Layer1.Thickness = 0.1e-3;
```

Import a top layer Gerber file to layer 2.

```
S.Layer2 = 'interdigital_Capacitor.gtl';
```

Create a PCBReader object using the stackUp object, S.

```
p = PCBReader('StackUp',S);
```

To update the Gerber file, convert the PCBReader object to a pcbComponent object.

```
pcbcapacitor = pcbComponent(p);
pcbcapacitor.FeedDiameter = 0.001
```

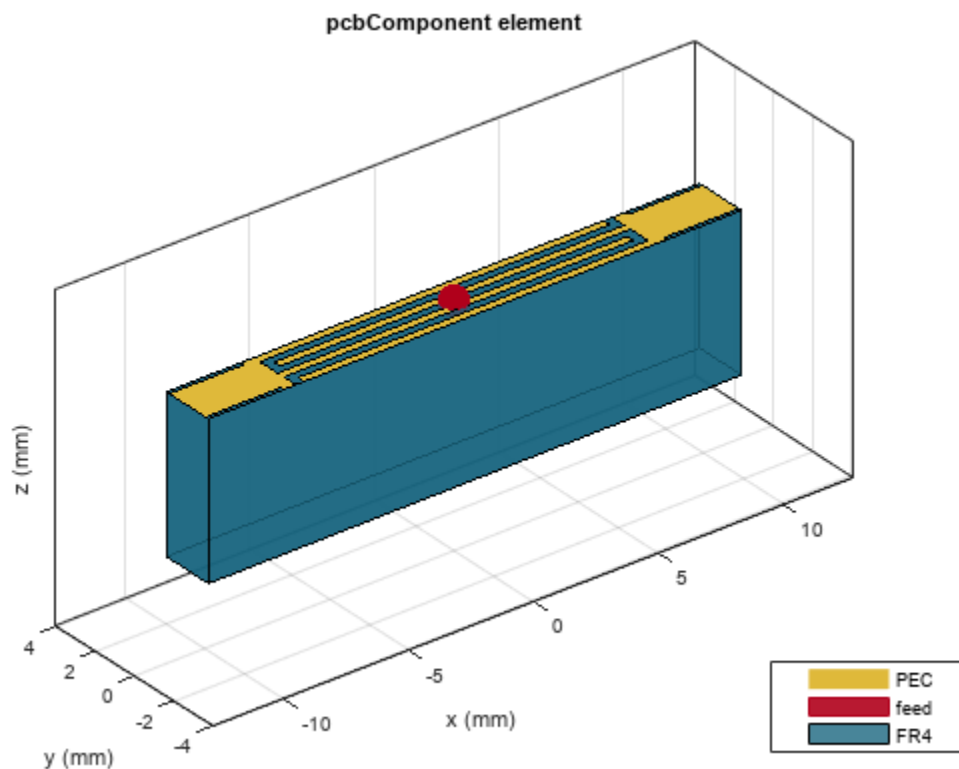
```
pcbcapacitor =
    pcbComponent with properties:
```

```
        Name: 'interdigital_Capacitor'
        Revision: 'v1.0'
        BoardShape: [1x1 antenna.Rectangle]
```

```
BoardThickness: 0.0061
  Layers: {1x4 cell}
FeedLocations: [0 0 2]
FeedDiameter: 1.0000e-03
ViaLocations: []
ViaDiameter: []
FeedViaModel: 'square'
  Conductor: [1x1 metal]
    Tilt: 0
    TiltAxis: [0 0 1]
    Load: [1x1 lumpedElement]
```

View the PCB component in the Gerber file.

```
show(pcbcapacitor)
```



Version History

Introduced in R2021b

See Also

[PCBReader](#) | [gerberRead](#) | [DielectricCatalog](#) | [dielectric](#)

pcbComponent

Create single or multifeed PCB component

Description

Use the `pcbComponent` object to create a multiport PCB component consisting of metal and dielectric layers.

Creation

Syntax

```
pcb = pcbComponent
pcb = pcbComponent(Name=Value)
```

Description

`pcb = pcbComponent` creates a two-port PCB component.

`pcb = pcbComponent(Name=Value)` sets “Properties” on page 1-213 using one or more name-value arguments. For example, `pcb = pcbComponent(Name=PCBWilkinson)` creates a PCB component named 'PCBWilkinson'. Properties not specified retain their default values.

Properties

Name — Name of PCB component

'MyPCB' (default) | character vector | string scalar

Name of the PCB component, specified as a character vector or string scalar.

Example: `component = pcbComponent(Name='PCBsplitter')`

Data Types: `char` | `string`

Revision — Revision details

'v1.0' (default) | character vector | string scalar

Design revision details of the PCB component, specified as a character vector or string scalar.

Example: `component = pcbComponent(Revision='v2.0')`

Data Types: `char` | `string`

BoardShape — Shape of PCB

`traceRectangular` (default) | object

Shape of the PC board, specified as a shape object. You can specify any one of the shapes from “Custom Geometry and PCB Fabrication”.

Example: `trace = tracerectangular; component = pcbComponent(BoardShape=trace)` creates a rectangle shaped trace on a PCB.

Data Types: `char` | `string`

BoardThickness — Height of PCB component

`0.0016` (default) | positive scalar

Height of the PCB component, specified as a positive scalar in meters. To understand more about **BoardThickness**, see “Board Thickness versus Dielectric Thickness in PCB”.

Example: `component = pcbComponent(BoardThickness=0.0026)`

Data Types: `double`

Layers — Metal and dielectric layers

`{[1×1 traceRectangular] [1×1 dielectric] [1×1 traceRectangular]}` (default) | cell array of metal and dielectric layers

Metal and dielectric layers, specified a cell array of metal and dielectric layers. You can specify one metal shape or one dielectric substrate per layer starting with the top layer and proceeding downward.

Data Types: `cell`

FeedLocations — Feed locations on PCB component

`[2×4 double]` (default) | *N*-by-3 array | *N*-by-4 array

Feed locations on the PCB component in Cartesian coordinates, specified as either an *N*-by-3 or *N*-by-4 array with *N* representing the number for ports on the PCB component. You can place the feed inside the board or at the edge of the board. The arrays translate into the following:

- *N*-by-3 - `[x, y, Layer]`
- *N*-by-4 - `[x, y, SigLayer, GndLayer]`

Example: `component = pcbComponent(FeedLocations=[-0.0187 0 1 2])`

Data Types: `double`

FeedDiameter — Diameter of center pin of feed connector

`0.0025` (default) | positive scalar

Diameter of center pin of the feed connector, specified as a positive scalar in meters.

Example: `component = pcbComponent(FeedDiameter=2.000e-04)`

Data Types: `double`

ViaLocations — Electrical short locations on PCB component

real vector of size *M*-by-4 array

Electrical short locations on the PCB component in Cartesian coordinates, specified as a real vector of size *M*-by-4 array. The array translates into the following:

- *M*-by-4 - `[x, y, SigLayer, GndLayer]`

Example: `component = pcbComponent(ViaLocations=[0 -0.025 1 2])`

Data Types: `double`

ViaDiameter – Diameter of electrical shorting pin used between metal layers

positive scalar | positive vector

Diameter of electrical shorting pin used between metal layers, specified as a positive scalar in meters for a single pin or a positive vector in meters for multiple pins. Number of values specified in this property must match the number of pins.

Example: `component = pcbComponent(ViaDiameter=1.0e-3)`

Data Types: double

FeedViaModel – Model for approximating feed and via

'strip' (default) | 'square' | 'hexagon' | 'octagon'

Model for approximating the feed and via, specified as one of the following:

- 'strip' - A rectangular strip approximation to the feed and via cylinder. This approximation is the simplest and results in a small mesh.
- 'square' - A four-sided polyhedron approximation to the feed and via cylinder.
- 'hexagon' - A six-sided polyhedron approximation to the feed and via cylinder.
- 'octagon' - A eight-sided polyhedron approximation to the feed and via cylinder.

Example: `component = pcbComponent(FeedViaModel='octagon')`

Data Types: char | string

Conductor – Type of metal material

'PEC' (default) | metal object

Type of the metal used as a conductor, specified as a metal object. You can choose any metal from the `MetalCatalog` or specify a metal of your choice. For more information, see `metal`. For more information on metal conductor meshing, see “Method of Moments Solver for Metal and Dielectric Structures”.

Example: `m = metal('Copper'); component = pcbComponent(Conductor=m)`

Load – Lumped elements

[1x1 LumpedElement] (default) | lumped element object handle

Lumped elements added to the PCB component feed, specified as a lumped element object handle. For more information, see `lumpedElement` Antenna Toolbox™.

Example: `Load = lumpedelement`. `lumpedelement` is the object handle for the load created using `lumpedElement`.

Tilt – Tilt angle of PCB component along Z-axis

0 (default) | scalar

Tilt angle of the PCB component along Z-axis, specified as a scalar or vector with each element unit in degrees.

Example: `Tilt=90`

Example: `pcb.Tilt = 90`

Data Types: double

TiltAxis – Tilt axis of PCB component

[0 0 1] (default) | 'Z'

Tilt axis of the PCB component, specified as [0 0 1] or 'Z'.

Example: `pcb.TiltAxis = 'Z'`

Data Types: double

Object Functions

current	Calculate and plot current distribution
charge	Calculate and plot charge distribution
feedCurrent	Calculate current at feed port
gerberWrite	Generate Gerber files
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
meshconfig	Change mesh mode of PCB component or shape structure
sparameters	Calculate S-parameters for RF PCB objects
show	Display PCB component structure or PCB shape

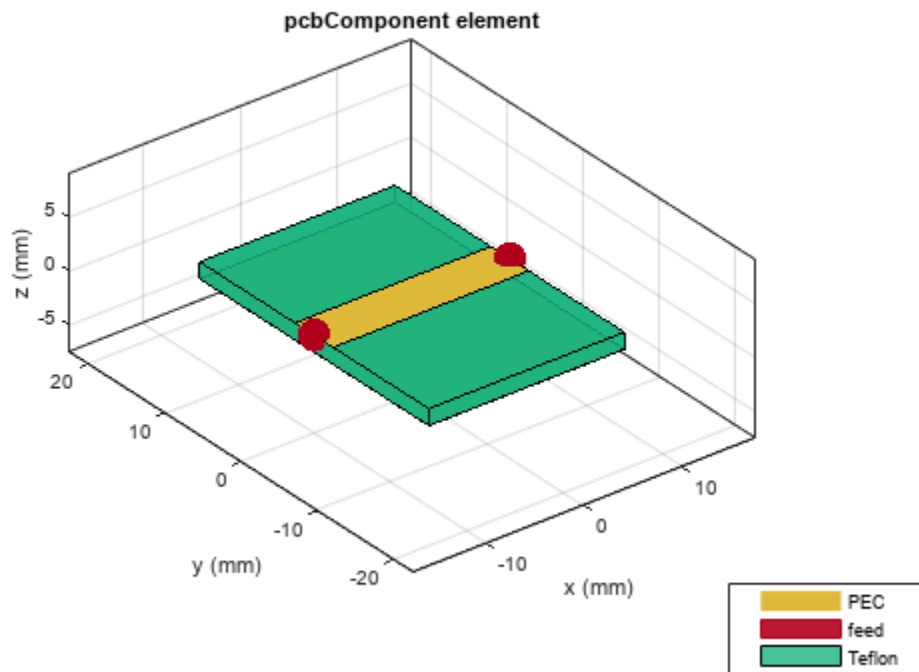
Examples**Create Default PCB Component and Plot S-Parameters**

Create a PCB component using default properties.

```
pcb = pcbComponent;
```

View the PCB component.

```
show(pcb)
```

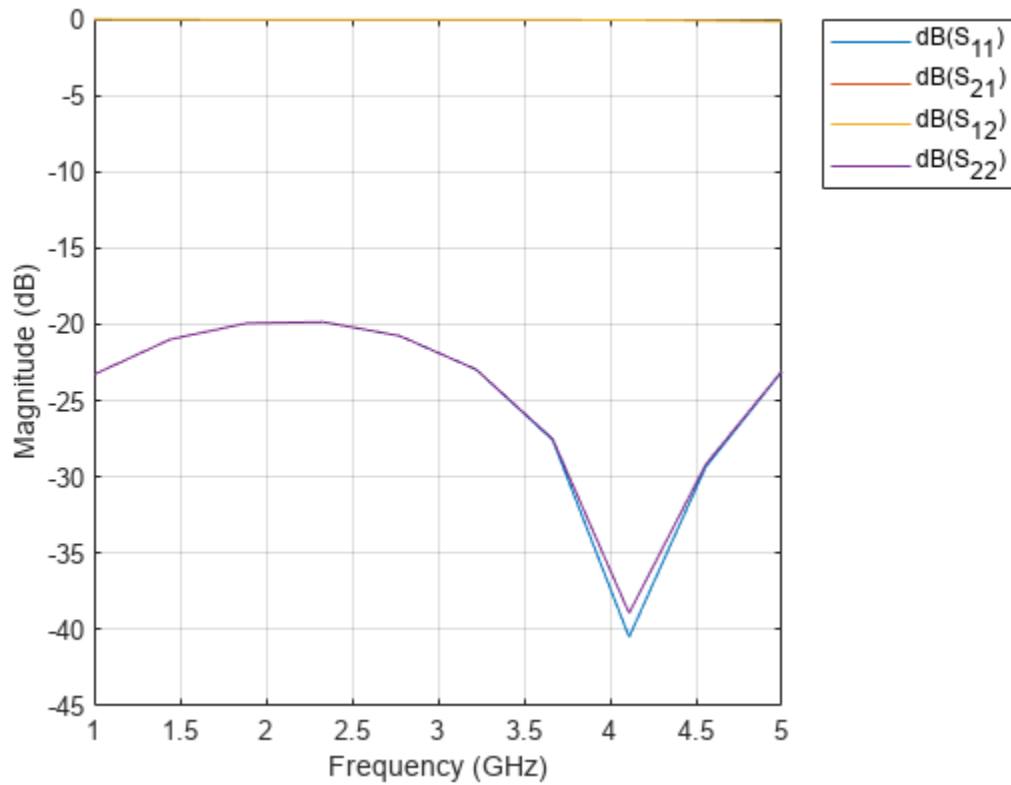


Calculate S-parameters over 10 frequencies from 1-5 GHz.

```
s=sparameters(pcb,linspace(1e9,5e9,10));
```

Plot the S-parameters.

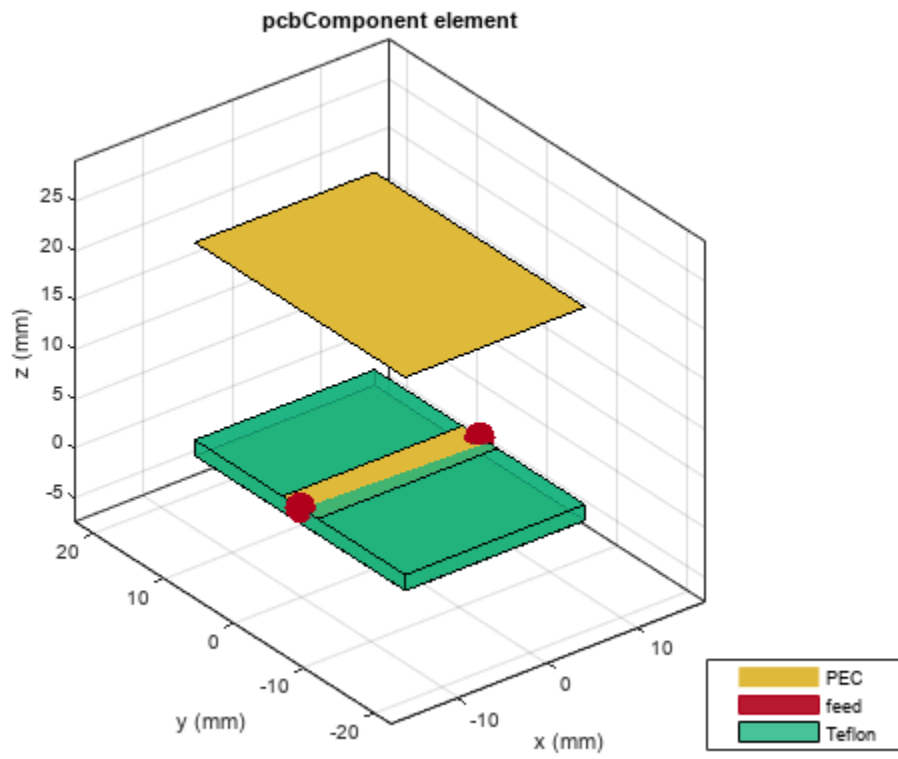
```
rfplot(s)
```



Create PCB Component with Lid on Top

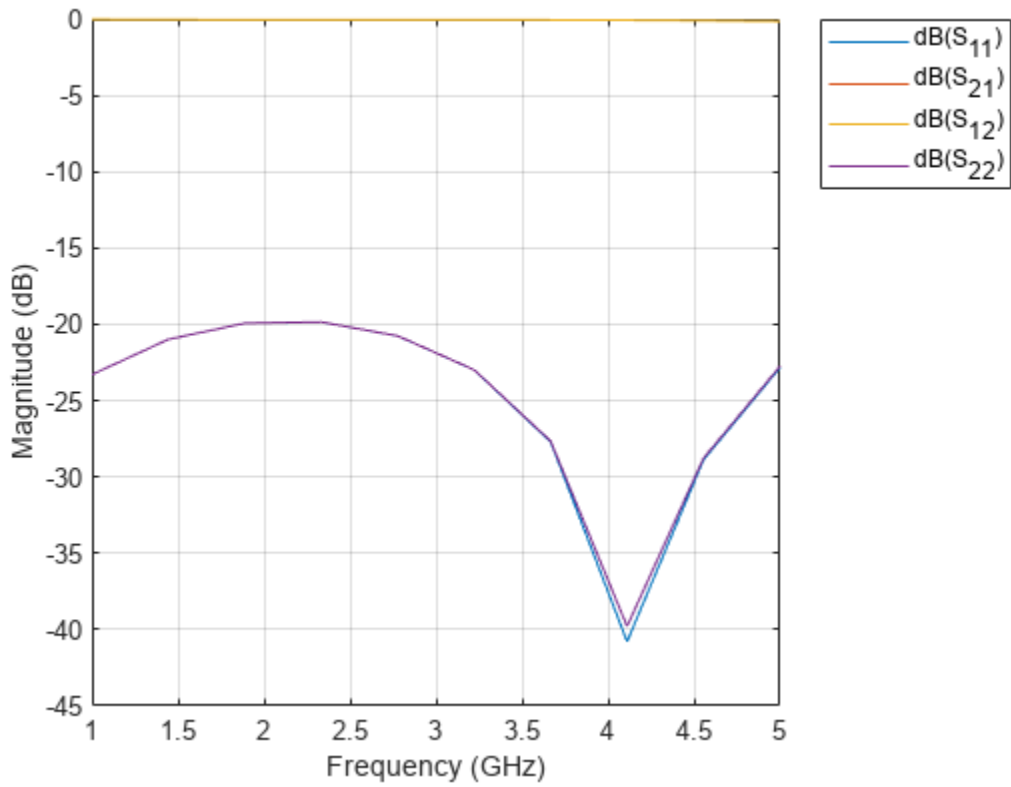
Create a PCB component with a lid at a distance of 2 cm above the component.

```
p = pcbComponent;  
pcblid = traceRectangular(Length=p.Layers{1}.Length,Width=p.Layers{3}.Width);  
dAir = dielectric('Air');  
dAir.Thickness = 2e-2;  
p.BoardThickness = p.BoardThickness + dAir.Thickness;  
p.Layers = {pcblid,dAir,p.Layers{1},p.Layers{2},p.Layers{3}};  
p.FeedLocations(:,3:4) = [3 5;3 5];  
show(p)
```



Calculate the S-parameters over the 10 frequencies from 1-5 GHz.

```
s = sparameters(p, linspace(1e9,5e9,10));  
rfplot(s)
```



Version History

Introduced in R2021b

See Also

[gerberRead](#) | [gerberWrite](#) | [PCBReader](#) | [PCBWriter](#) | [PCBServices](#)

pcbElement

Create RF Toolbox circuit element

Description

Use the `pcbElement` object to create an RF Toolbox circuit element.

Creation

Syntax

```
circuit_element = pcbElement(rfpcbobject)
circuit_element = pcbElement(rfpcbobject,Name=Value)
```

Description

`circuit_element = pcbElement(rfpcbobject)` creates a PCB element object from a PCB component. You can use this element in an RF Toolbox circuit.

`circuit_element = pcbElement(rfpcbobject,Name=Value)` sets properties using one or more name-value arguments.

Input Arguments

rfpcbobject — PCB component object

RF PCB object

PCB component object, specified as an RF PCB object. For a complete list of the PCB components, see “PCB Components Catalog”.

Properties

Behavioral — Computes S-parameters using behavioral model

'true' (default) | 'false'

Compute S-parameters of the PCB element using the behavioral model, specified as a logical true or false. When you specify true, the object calculates the S-parameters using the behavioral model. When you specify false, the object calculates the S-parameters using the full-wave solver. For components and shapes that support the behavioral model, see “Behavioral Models” and `sparameters`.

Data Types: `logical`

Examples

Calculate S-Parameters of Two Capacitors in Circuit

Create a circuit using default properties.

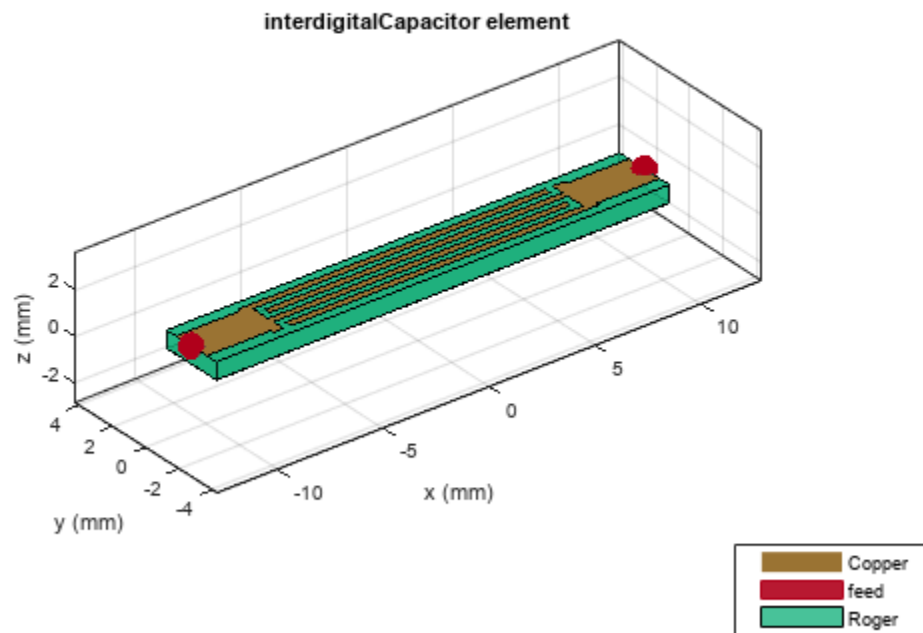
```
ckt = circuit;
```

Create two interdigital capacitors, one using default properties and one with three fingers.

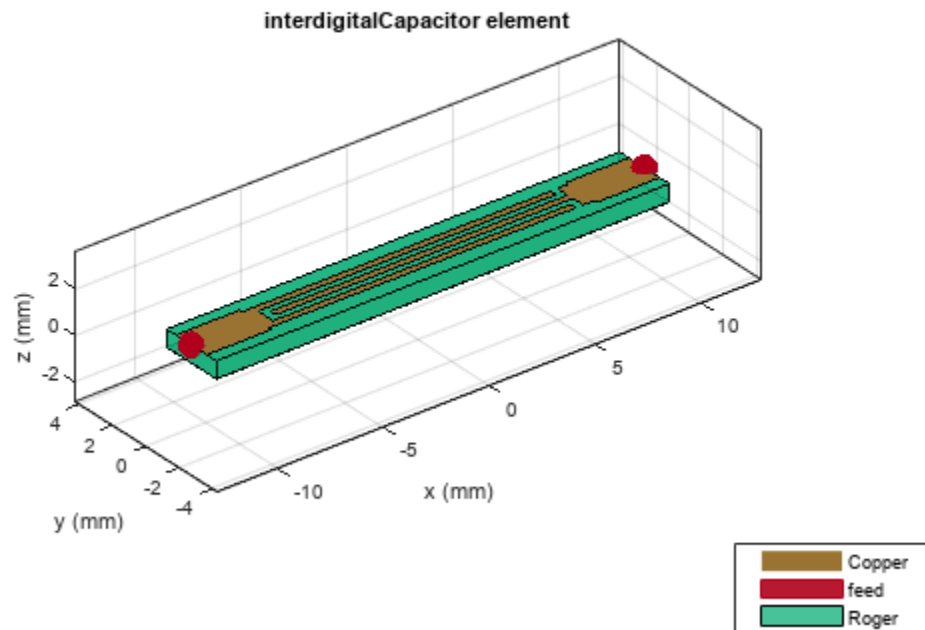
```
c1 = interdigitalCapacitor;  
c2 = interdigitalCapacitor('NumFingers',3);
```

View both c1 and c2.

```
show(c1)
```



```
figure;  
show(c2)
```



Convert c2 to a PCB element with the Behavioral property set to false.

```
p = pcbElement(c2,'Behavioral',false);
```

Add both capacitors to the circuit object.

```
add(ckt,[1 2 0 0],c1) % default pcbElement created automatically
add(ckt,[2 3 0 0],p)
setports(ckt,[1 0],[3 0])
```

Calculate the S-parameters.

```
S = sparameters(ckt,8e9)
```

```
S =
sparameters: S-parameters object
```

```
    NumPorts: 2
    Frequencies: 8.0000e+09
    Parameters: [2x2 double]
    Impedance: 50
```

rfparam(obj,i,j) returns S-parameter S_{ij}

Version History

Introduced in R2021b

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

sparameters

Topics

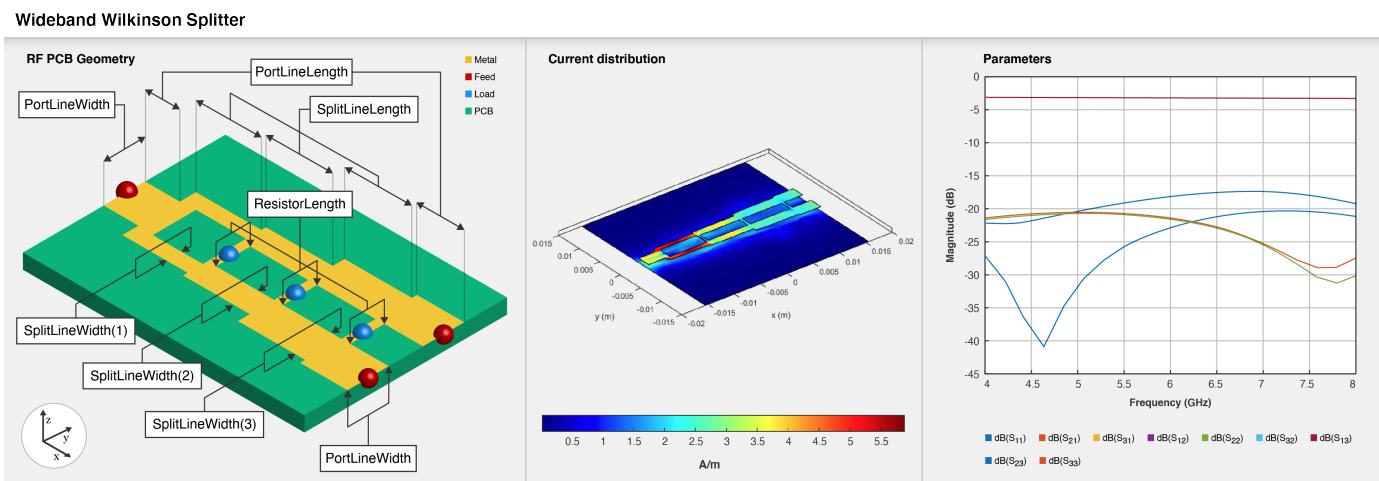
“Behavioral Models”

wilkinsonSplitterWideband

Create wideband Wilkinson power divider

Description

Use the `wilkinsonSplitterWideband` object to create a wideband Wilkinson power divider. The wide bandwidth is achieved through the multiple sections that are used in the construction of the divider. It is a lossless power divider and provides matching at all ports. The isolation between the output ports is achieved using a resistor connected in between the output ports.



Creation

Syntax

```
splitter = wilkinsonSplitterWideband
splitter = wilkinsonSplitterWideband(Name=Value)
```

Description

`splitter = wilkinsonSplitterWideband` creates a wideband Wilkinson splitter with a Teflon substrate. The default properties are for a resonating frequency of 6 GHz.

`splitter = wilkinsonSplitterWideband(Name=Value)` sets "Properties" on page 1-117 using one or more name-value arguments. For example, `wilkinsonSplitterWideband(PortLineLength=0.0300)` creates a wideband Wilkinson splitter with an input and output line length of 0.0300 meters. Properties not specified retain their default values.

Properties

Shape — Shape of sections

"Rectangular" (default) | "Circular"

Shape of the sections, specified as "Rectangular" or "Circular".

Example: `splitter = wilkinsonSplitterWideband(Shape="Circular")`

Data Types: char | string

NumSections — Number of sections

3 (default) | positive scalar

Number of sections, specified as a positive scalar. The minimum value is 2 and the maximum value is 7.

Example: `splitter = wilkinsonSplitterWideband(NumSections=4)`

Data Types: double

PortLineLength — Length of input and output line

0.0040 (default) | positive scalar

Length of the input and the output line in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitterWideband(PortLineLength=0.0070)`

Data Types: double

PortLineWidth — Width of input and output line

0.0024 (default) | positive scalar

Width of the input and the output line in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitterWideband(PortLineWidth=0.0034)`

Data Types: double

SplitLineLength — Length of quarter wave transformer

0.0080 (default) | positive scalar

Length of the quarter wave transformer in meters, specified as a positive scalar. The typical length of a Wilkinson splitter is $\lambda/4$.

Example: `splitter = wilkinsonSplitterWideband(SplitLineLength=0.0570)`

Data Types: double

SplitLineWidth — Width of quarter wave transformer

[8.5495e-04 0.0014 0.0021] (default) | two-element vector

Width of the quarter wave transformer in meters, specified as a two-element vector of positive elements.

Example: `splitter = wilkinsonSplitterWideband(SplitLineWidth=[0.00780 0.00890])`

Data Types: double

ResistorLength — Length of resistor

0.0020 (default) | positive scalar

Length of the resistor in meters, specified as a positive scalar. The resistor length decided the spacing between the output ports.

Example: `splitter = wilkinsonSplitterWideband(ResistorLength=0.0050)`

Data Types: double

Resistance — Resistance value

[100 183.4008 141.4214] (default) | three-element vector

Resistance value in ohms, specified as a three-element vector of positive elements. The default value is for an equal-split wideband Wilkinson splitter.

Example: `splitter = wilkinsonSplitterWideband(Resistance=[90 173.4008 166.4214])`

Data Types: double

Height — Height of splitter from ground plane

7.6200e-04 (default) | positive scalar

Height of the splitter from the ground plane in meters, specified as a positive scalar. If your substrate has multiple layers, you can use the Height property to create a wideband Wilkinson splitter where the two dielectrics interface.

Example: `splitter = wilkinsonSplitterWideband(Height=0.0076)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0300 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `splitter = wilkinsonSplitterWideband(GroundPlaneWidth=0.046)`

Example: double

Substrate — Type of dielectric material

'Teflon' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. The thickness of the default dielectric material Teflon is 0.0016 m or the same value as the Height property.

Example: `d = dielectric("FR4"); splitter = wilkinsonSplitterWideband(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'PEC' (default) | metal object

Type of metal used in the conducting layers, specified as a metal object.

Example: `m = metal("Copper"); splitter = wilkinsonSplitterWideband(Conductor=m)`

Data Types: string | char

Object Functions

charge	Calculate and plot charge distribution
current	Calculate and plot current distribution
design	Design wideband Wilkinson splitter around specified frequency
feedCurrent	Calculate current at feed port
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples

Default Wideband Wilkinson Splitter

Create a wideband Wilkinson splitter with default properties.

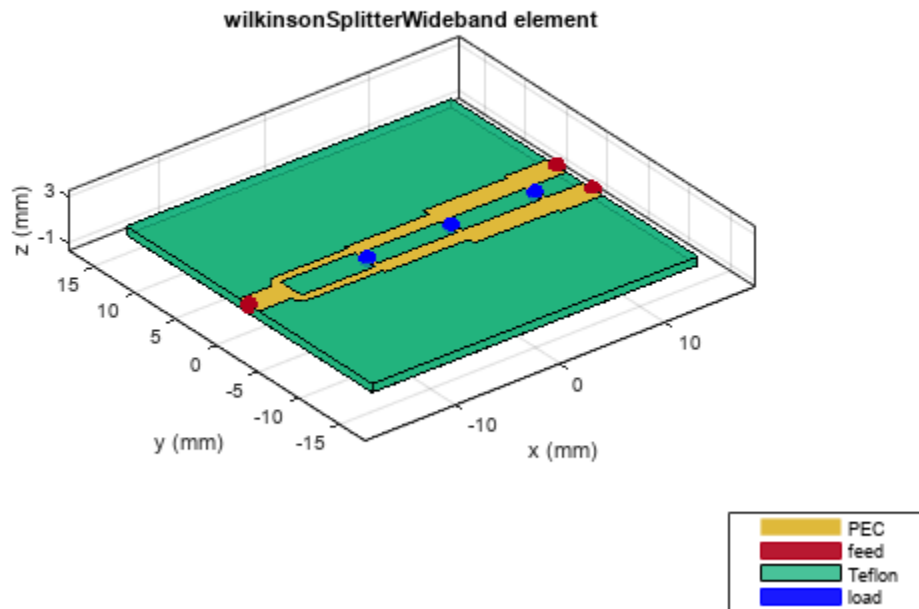
```
splitter = wilkinsonSplitterWideband

splitter =
  wilkinsonSplitterWideband with properties:

        Shape: 'Rectangular'
      NumSections: 3
    PortLineLength: 0.0040
    PortLineWidth: 0.0024
    SplitLineLength: 0.0080
    SplitLineWidth: [8.5495e-04 0.0014 0.0021]
    ResistorLength: 0.0020
        Resistance: [100 183.4008 141.4214]
            Height: 7.6200e-04
    GroundPlaneWidth: 0.0300
        Substrate: [1x1 dielectric]
        Conductor: [1x1 metal]
```

Visualize the splitter.

```
show(splitter);
```

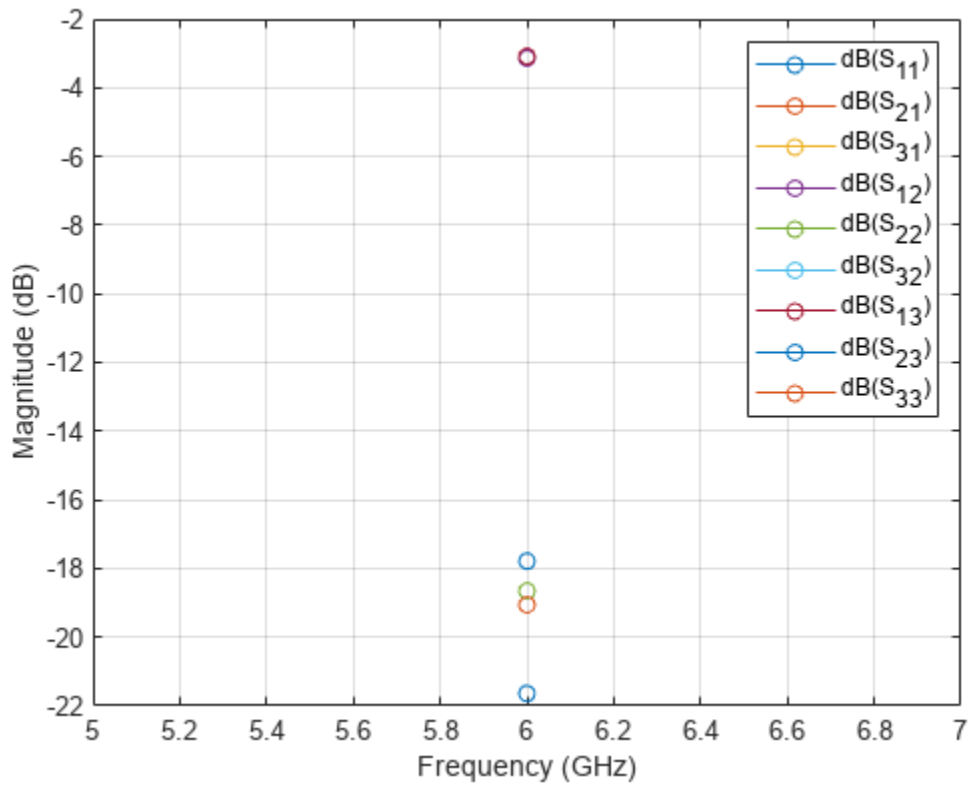
S-Parameters of Wideband Wilkinson

Create a wideband Wilkinson splitter with default properties.

```
splitter = wilkinsonSplitterWideband;
```

Calculate the s-parameters of this splitter at 6 GHz.

```
spar = sparameters(splitter,6e9);  
figure;  
rfplot(spar);
```



Wideband Wilkinson with Multi-Layered Dielectric

Create a wideband Wilkinson splitter with the default properties.

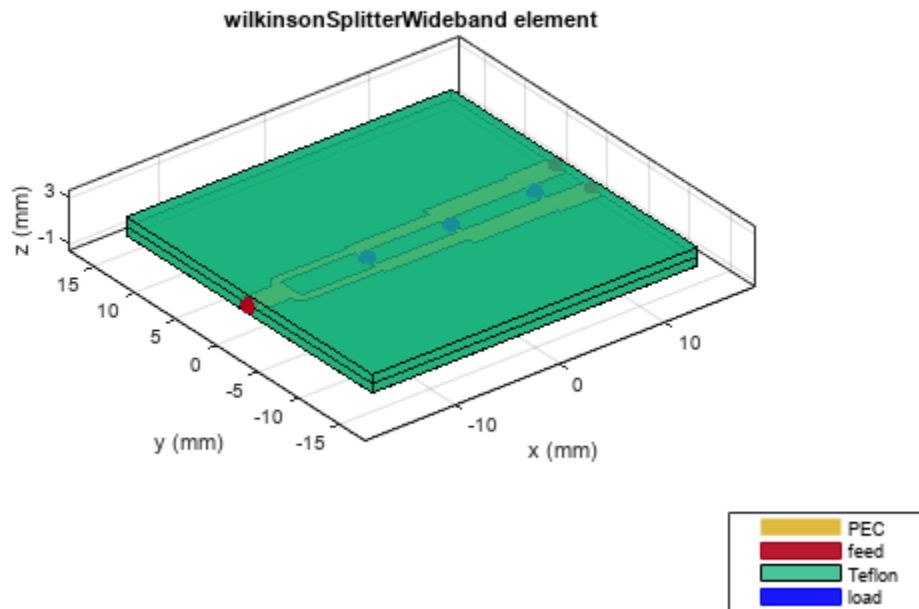
```
splitter = wilkinsonSplitterWideband;
```

Change the substrate to a multilayered substrate. Change the height of the splitter.

```
splitter.Substrate = dielectric('Name',{'Teflon','Teflon'},'EpsilonR', ...
    [2.1 2.1],'LossTangent',[0 0],'Thickness',[0.8e-3 0.8e-3]);
splitter.Height = 0.8e-3;
```

Visualize the splitter.

```
show(splitter);
```



Version History

Introduced in R2022a

References

- [1] Mishra, B, A.Rahman, S.Shaw, M.Mohammed, S.Mondal, P.P.Shankar. "Design of an ultra-wideband Wilkinson power divider." *Automation, Control, Energy, and Systems (IEEE 2014)*
- [2] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

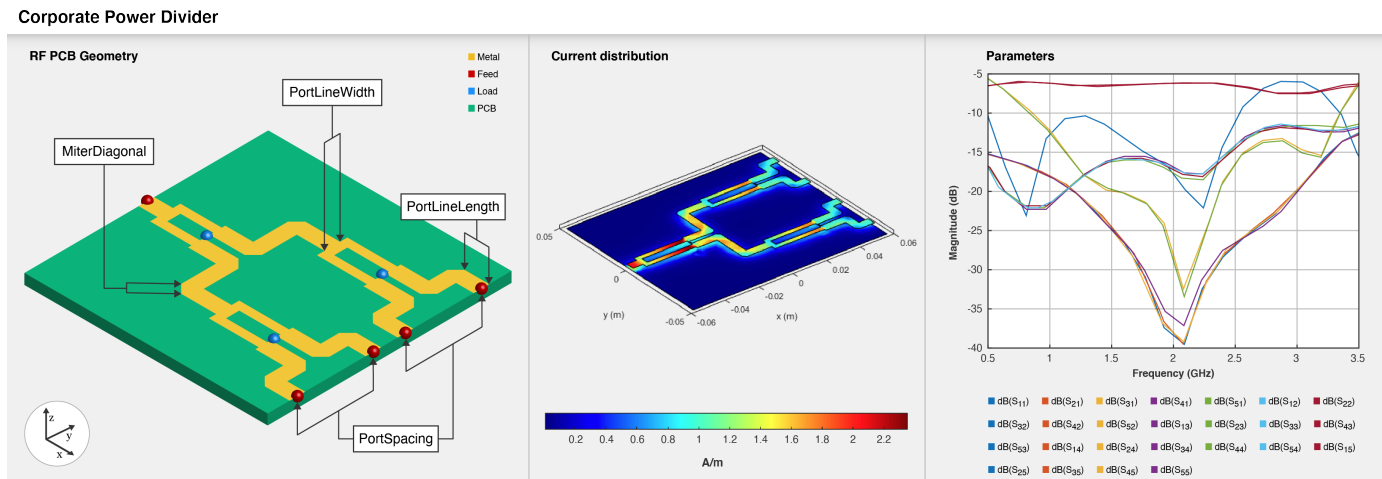
wilkinsonSplitter | wilkinsonSplitterUnequal

powerDividerCorporate

Create corporate power divider

Description

Use the `powerDividerCorporate` object to create a corporate power divider to divide power equally or unequally from a single port to multiple ports.



Creation

Syntax

```
divider = powerDividerCorporate
divider = powerDividerCorporate(Name=Value)
```

Description

`divider = powerDividerCorporate` creates a corporate power divider with default values. The default properties are for an operating frequency of 1.6 GHz.

`divider = powerDividerCorporate(Name=Value)` sets “Properties” on page 1-117 using one or more name-value arguments. For example, `powerDividerCorporate(PortLineLength=0.0300)` creates a corporate power divider with an input and output line length of 0.0300 meters. Properties not specified retain their default values.

Properties

NumOutputPorts — Number of output ports

4 (default) | 2 | 8 | 16 | 32 | positive scalar

Number of output ports, specified as a positive scalar.

Example: `divider = powerDividerCorporate(NumOutputPorts=8)`

Data Types: double

SplitterElement — Basic divider element

"wilkinsonSplitter (default) | "wilkinsonSplitterUnequal" |
"wilkinsonSplitterWideband"

Basic divider element, specified as `wilkinsonSplitter`, `wilkinsonSplitterUnequal`, or `wilkinsonSplitterWideband` object.

Example: `divider = powerDividerCorporate(SplitterElement="wilkinsonSplitterWideband")`

Data Types: char | string

Corner — Shape of corner at each bend

'Mitered' (default) | 'Curved'

Shape of corner at each bend specified as 'Mitered' or 'Curved'. If you set `Corner` to 'Mitered', specify the miter diagonal by setting the `MiterDiagonal` property. If you set `Corner` to 'Curved', specify the radius of the curve by setting the `CurveRadius` property.

Example: `divider = powerDividerCorporate(SplitterElement='Curved')`

Data Types: char | string

MiterDiagonal — Length of miter diagonal

0.0025 (default) | positive scalar

Length of the miter diagonal in meters, specified as a positive scalar. This property applies only when you set the `Corner` property to 'Mitered'.

Example: `divider = powerDividerCorporate(MiterDiagonal=0.0046)`

CurveRadius — Radius of curve

0.0025 (default) | positive scalar

Radius of the curve at the bends in meters, specified as a positive scalar. This property applies only when you set the `Corner` property to 'Curved'.

Example: `divider = powerDividerCorporate(CurveRadius=0.0046)`

Data Types: double

PortLineLength — Length of input and output line

0.0080 (default) | positive scalar

Length of the input and the output line in meters, specified as a positive scalar.

Example: `divider =powerDividerCorporate(PortLineLength=0.0070)`

Data Types: double

PortLineWidth — Width of input and output line

0.0049 (default) | positive scalar

Width of the input and the output line in meters, specified as a positive scalar.

Example: `divider =powerDividerCorporate(PortLineWidth=0.0034)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0960 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `divider =powerDividerCorporate(GroundPlaneWidth=0.046)`

Example: double

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>design</code>	Design corporate power divider around specified frequency
<code>feedCurrent</code>	Calculate current at feed port
<code>layout</code>	Plot all metal layers and board shape
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>shapes</code>	Extract all metal layer shapes of PCB component
<code>show</code>	Display PCB component structure or PCB shape
<code>sparameters</code>	Calculate S-parameters for RF PCB objects

Examples

Default Corporate Power Divider

Create a corporate power divider with default values.

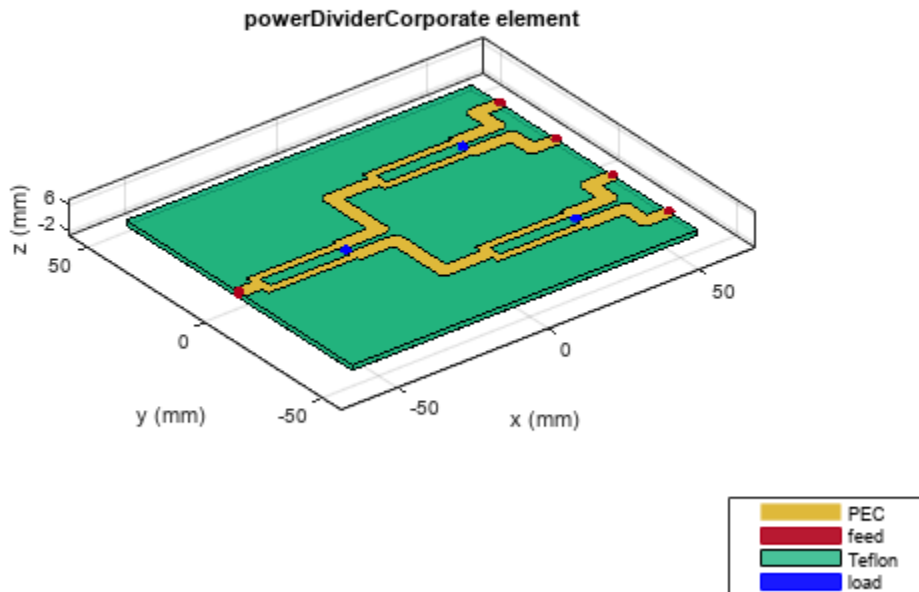
```
divider = powerDividerCorporate

divider =
  powerDividerCorporate with properties:

    NumOutputPorts: 4
    SplitterElement: [1x1 wilkinsonSplitter]
        Corner: 'Mitered'
    MiterDiagonal: 0.0025
    PortSpacing: 0.0240
    PortLineLength: 0.0080
    PortLineWidth: 0.0049
    GroundPlaneWidth: 0.0960
```

Visualize the power divider.

```
show(divider)
```



Corporate Power Divider with Multi-Layered Dielectric

Create a corporate power divider with default values.

```
divider = powerDividerCorporate;
```

Change the substrate of the splitter element to a multi-layered substrate.

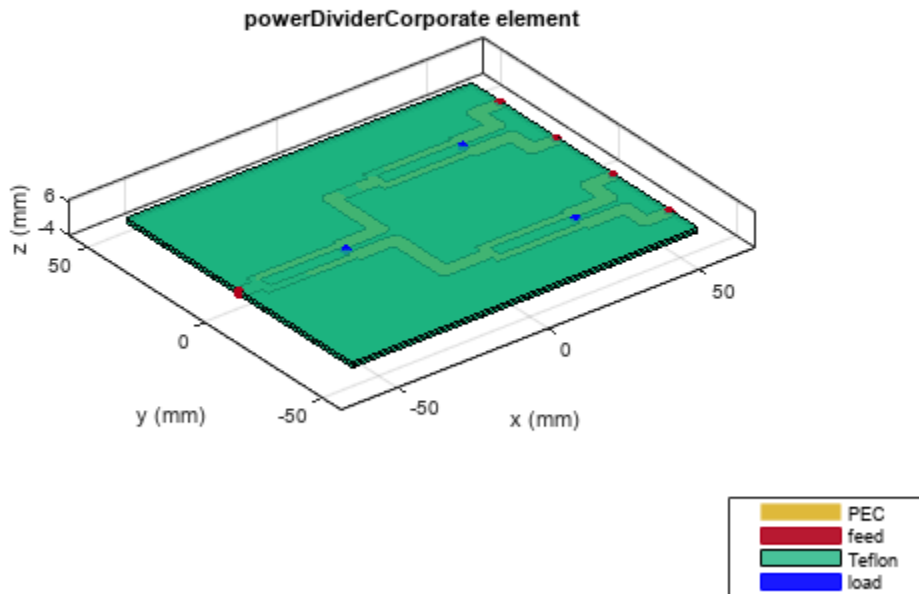
```
divider.SplitterElement.Substrate = dielectric('Name', {'Teflon', 'Teflon'}, 'EpsilonR', ...
    [2.1 2.1], 'LossTangent', [0 0], 'Thickness', [0.8e-3 0.8e-3]);
```

Change the height of the splitter element.

```
divider.SplitterElement.Height = 0.8e-3;
```

Visualize the power divider

```
show(divider)
```



Version History

Introduced in R2022a

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

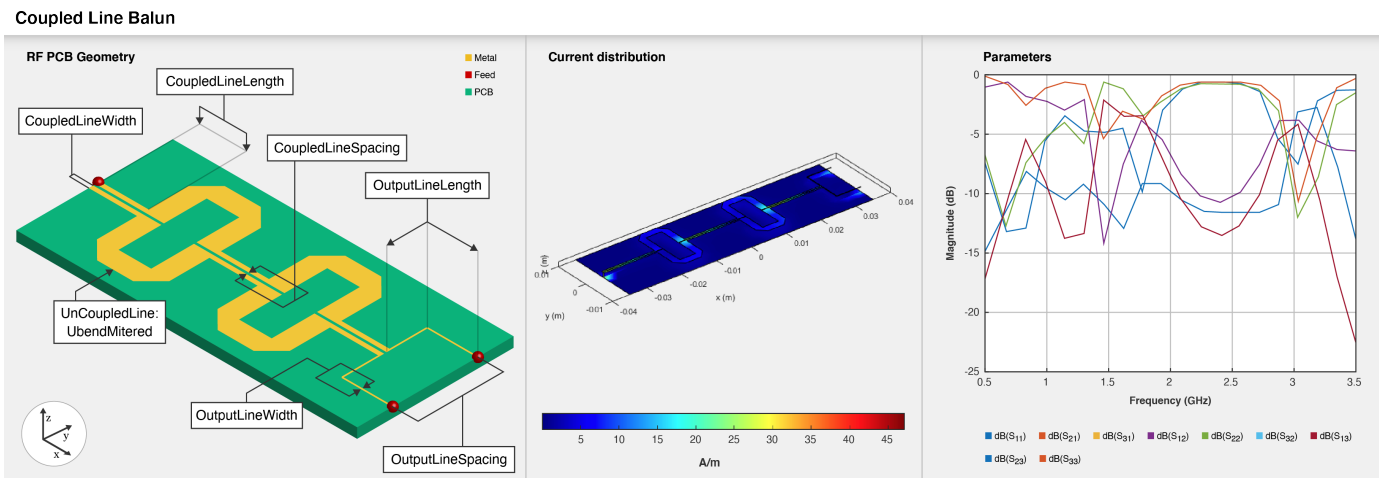
[wilkinsonSplitter](#) | [wilkinsonSplitterUnequal](#) | [wilkinsonSplitterWideband](#)

balunCoupledLine

Create multisection coupled-line balun on X-Y plane

Description

Use the `balunCoupledLine` object to create a multisection coupled-line balun with an unbalanced input and a balanced output. The output signal has a phase difference of 180 degrees.



Creation

Syntax

```
balun = balunCoupledLine
balun = balunCoupledLine(Name=Value)
```

Description

`balun = balunCoupledLine` creates a coupled-line balun in the microstrip form. The default properties are for a resonant frequency of 2.96 GHz.

`balun = balunCoupledLine(Name=Value)` sets “Properties” on page 1-42 using one or more name-value arguments. For example, `balunCoupledLine(OutputLineLength=0.0286)` creates a coupled-line balun with an output line length of 0.0286 meters. Properties not specified retain their default values.

Properties

NumCoupledLineSection — Number of coupled-line sections

3 (default) | positive scalar

Number of coupled-line sections, specified as a positive scalar. The minimum number of sections you can specify is two and the maximum number is five.

Example: `balun = balunCoupledLine(NumCoupledLine=4)`

Data Types: double

CoupledLineLength — Length of coupled line

0.01526 (default) | positive scalar

Length of the coupled line in meters, specified as a positive scalar.

Example: `balun = balunCoupledLine(CoupledLineLength=0.0254)`

Data Types: double

CoupledLineWidth — Width of coupled line

0.0004 (default) | positive scalar

Width of the coupled line in meters, specified as a positive scalar.

Example: `balun = balunCoupledLine(CoupledLineWidth=0.0005)`

Data Types: double

CoupledLineSpacing — Distance between coupled lines

0.00014 (default) | positive scalar

Spacing between the coupled lines in meters, specified as a positive scalar.

Example: `balun = balunCoupledLine(CoupledLineSpacing=0.00015)`

Data Types: double

UncoupledLineShape — Shape of uncoupled-line section

`ubendMitered` (default) | shape object

Shape of the uncoupled-line section, specified as a shape object. The default shape of the uncoupled-line section is the `ubendMitered` shape. The default dimensions of the `ubendMitered` shape are: length of [0.0082 0.00453 0.0082], width of [0.002 0.002 0.002], and a miter diagonal of 0.002828.

Example: `balun = balunCoupledLine(UncoupledLineShape=ubendMitered)`

Data Types: char | string

OutputLineLength — Length of quarter wave transformer

0.0124 (default) | positive scalar

Length of the quarter wave transformer used to extend the ports in meters, specified as a positive scalar.

Example: `balun = balunCoupledLine(OutputLineLength=0.0224)`

Data Types: double

OutputLineWidth — Width of output line

0.000153 (default) | positive scalar

Width of the output line in meters, specified as a positive scalar.

Example: `balun = balunCoupledLine(OutputLineWidth=0.000253)`

Data Types: double

OutputLineSpacing — Distance between output ports

0.011 (default) | positive scalar

Spacing between the output ports in meters, specified as a positive scalar.

Example: `balun = balunCoupledLine(OutputLineSpacing=0.022)`

Data Types: double

Height — Height of coupled-line balun from ground plane

0.0013 (default) | positive scalar

Height of the coupled-line balun from the ground plane in meters, specified as a positive scalar.

Example: `balun = balunCoupledLine(Height=0.022)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0200 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `balun = balunCoupledLine(GroundPlaneWidth=0.032)`

Example: double

Substrate — Type of dielectric material

'FR4' (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. The default height of the substrate is 0.0013 meters.

Example: `d = dielectric("RTDuriod"); balun = balunCoupledLine(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'PEC' (default) | metal object

Type of metal used in the conducting layers, specified as a metal object.

Example: `m = metal("Copper"); balun = balunCoupledLine(Conductor=m)`

Data Types: string | char

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>designCoupledLine</code>	Calculate dimensions of coupled-line section for specified frequency
<code>designUncoupledLine</code>	Calculate dimensions of uncoupled-line section for specified frequency
<code>designOutputLine</code>	Calculate dimensions of output line section for specified frequency
<code>feedCurrent</code>	Calculate current at feed port
<code>layout</code>	Plot all metal layers and board shape

mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples

Default Coupled-Line Balun

Create a coupled-line balun using default properties.

```
balun = balunCoupledLine
```

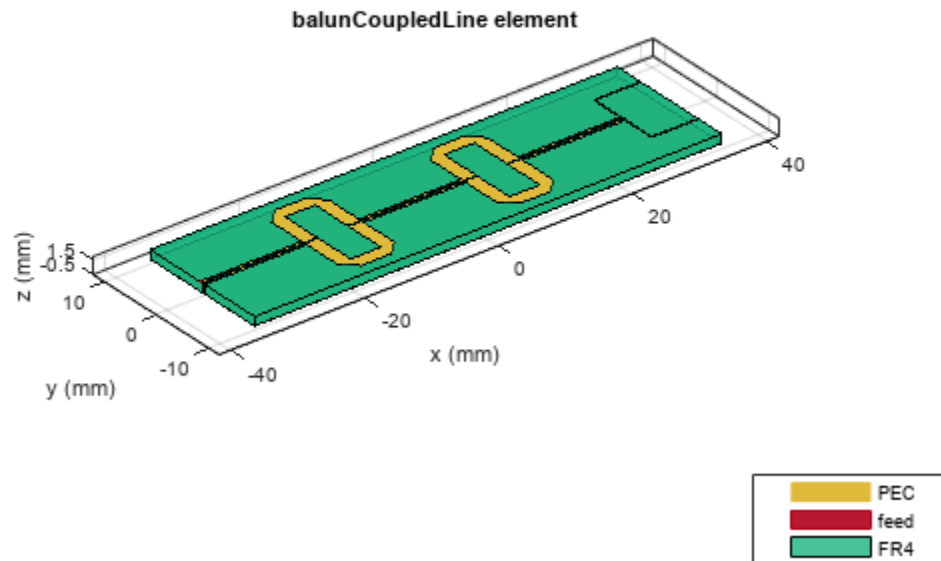
```
balun =
```

```
  balunCoupledLine with properties:
```

```
    NumCoupledLineSection: 3
      CoupledLineLength: 0.0153
      CoupledLineWidth: 4.0000e-04
      CoupledLineSpacing: 1.4000e-04
      UncoupledLineShape: [1x1 ubendMitered]
      OutputLineLength: 0.0124
      OutputLineWidth: 1.5300e-04
      OutputLineSpacing: 0.0110
      Height: 0.0013
      GroundPlaneWidth: 0.0200
      Substrate: [1x1 dielectric]
      Conductor: [1x1 metal]
```

Visualize the coupled-line balun.

```
show(balun)
```



Coupled-Line Balun with Multiple Dielectric Layers

Create a coupled-line balun using default properties.

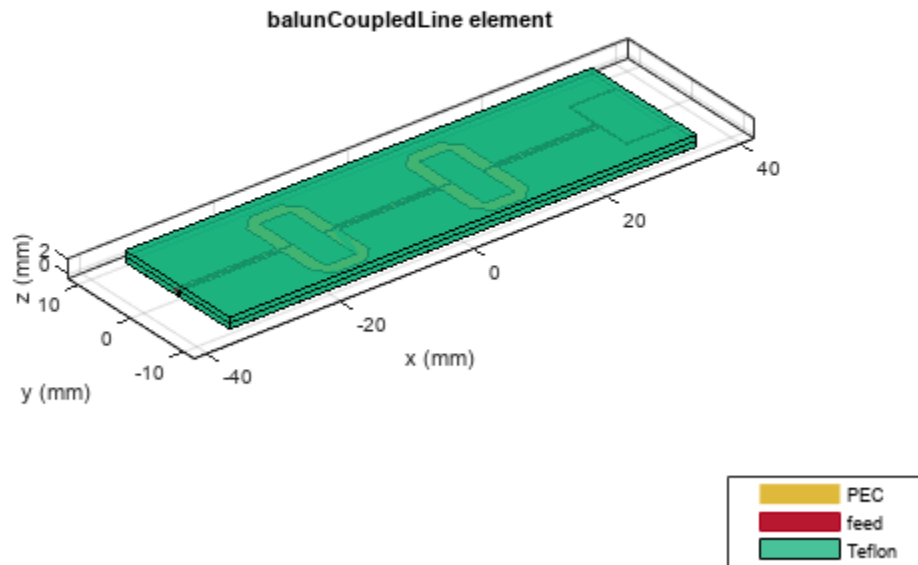
```
balun = balunCoupledLine;
```

Change the substrate and the dielectric of the balun.

```
balun.Substrate = dielectric('Name',{'Teflon','Teflon'},'EpsilonR', ...
    [2.1 2.1],'LossTangent',[0 0],'Thickness',[0.8e-3 0.8e-3]);
balun.Height = 0.8e-3;
```

Visualize the balun.

```
show(balun)
```



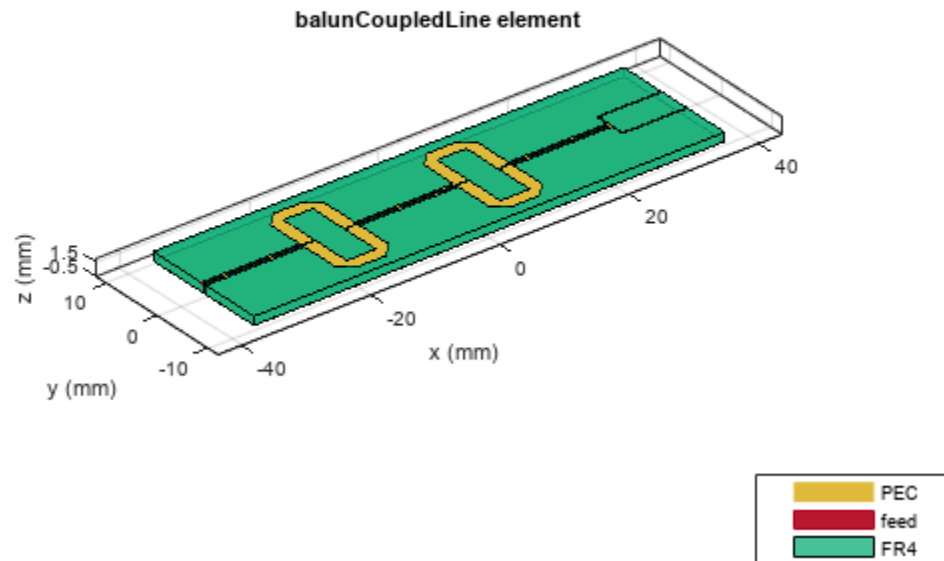
Customize Coupled-Line Balun

Create a coupled-line balun with an `OutputLineSpacing` of 5 mm.

```
balun = balunCoupledLine('OutputLineSpacing',0.005);
```

Visualize the balun.

```
show(balun);
```



Design Coupled Line Balun at 4 GHz

Define the frequency at 4 GHz.

```
f = 4e9;
```

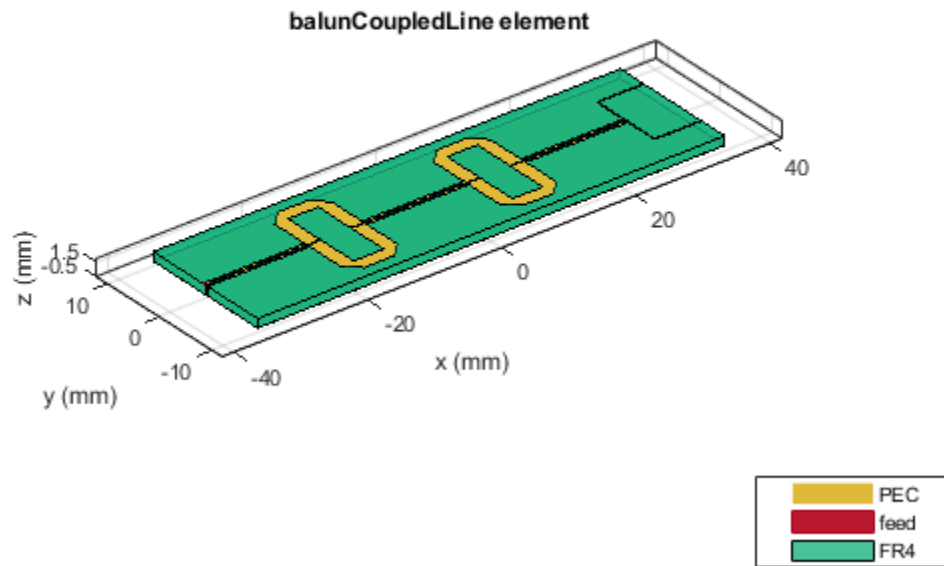
Create a coupled line balun object.

```
balun = balunCoupledLine
```

```
balun =  
  balunCoupledLine with properties:
```

```
  NumCoupledLineSection: 3  
    CoupledLineLength: 0.0153  
      CoupledLineWidth: 4.0000e-04  
    CoupledLineSpacing: 1.4000e-04  
  UncoupledLineShape: [1x1 ubendMitered]  
    OutputLineLength: 0.0124  
      OutputLineWidth: 1.5300e-04  
    OutputLineSpacing: 0.0110  
      Height: 0.0013  
    GroundPlaneWidth: 0.0200  
      Substrate: [1x1 dielectric]  
      Conductor: [1x1 metal]
```

```
show(balun)
```



Step 1: Design coupled line section

Design the coupled line section of the balun with an even mode impedance of 159 ohms and an odd mode impedance of 51 ohms. Use the helper function **designCoupledLine**.

```
[ClineL,ClineW,ClineS] = designCoupledLine(balun,f,'Z0e',159,'Z0o',51)
```

```
ClineL = 0.0107
```

```
ClineW = 4.2682e-04
```

```
ClineS = 1.4374e-04
```

Step 2: Design uncoupled line section

Design the uncoupled line section of the balun with the even and odd mode impedance of 59 ohms. Use the helper function **designUncoupledLine**.

```
[unclineL,unclineW] = designUncoupledLine(balun,f,'Z0',59,'LineLength',0.25)
```

```
unclineL = 0.0103
```

```
unclineW = 0.0018
```


Step 3: Design output line section

Design the output line section of the balun at the same frequency to extend the port 2 and port3. Use the helper function **designOutputLine**.

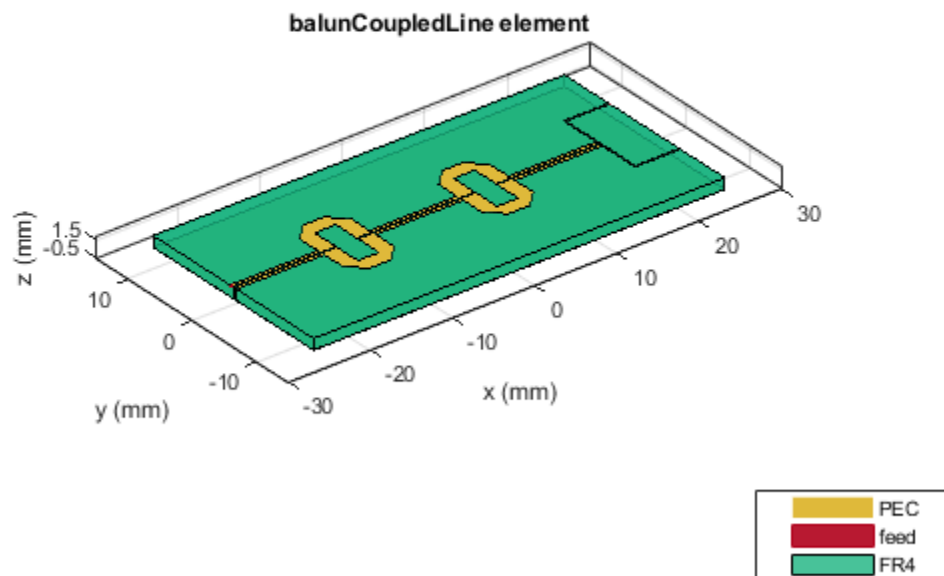
```
[OutL,OutW] = designOutputLine(balun,f,'Z0e',159,'Z0o',51,'Z0',59,'Zref',50)
```

```
OutL = 0.0109
```

```
OutW = 1.6115e-04
```

Set all the design dimensions to the coupled balun object.

```
balun.CoupledLineLength = ClineL;
balun.CoupledLineWidth = ClineW;
balun.CoupledLineSpacing = ClineS;
UncoupledLine = ubendMitered;
UncoupledLine.Length = [unclineL/2,unclineL/4,unclineL/2];
UncoupledLine.Width = [unclineW,unclineW,unclineW];
balun.UncoupledLineShape = UncoupledLine;
balun.OutputLineLength = OutL;
balun.OutputLineWidth = OutW;
balun.OutputLineSpacing = OutL+ClineS;
gndW = 25e-3;
balun.GroundPlaneWidth = gndW;
show(balun)
```

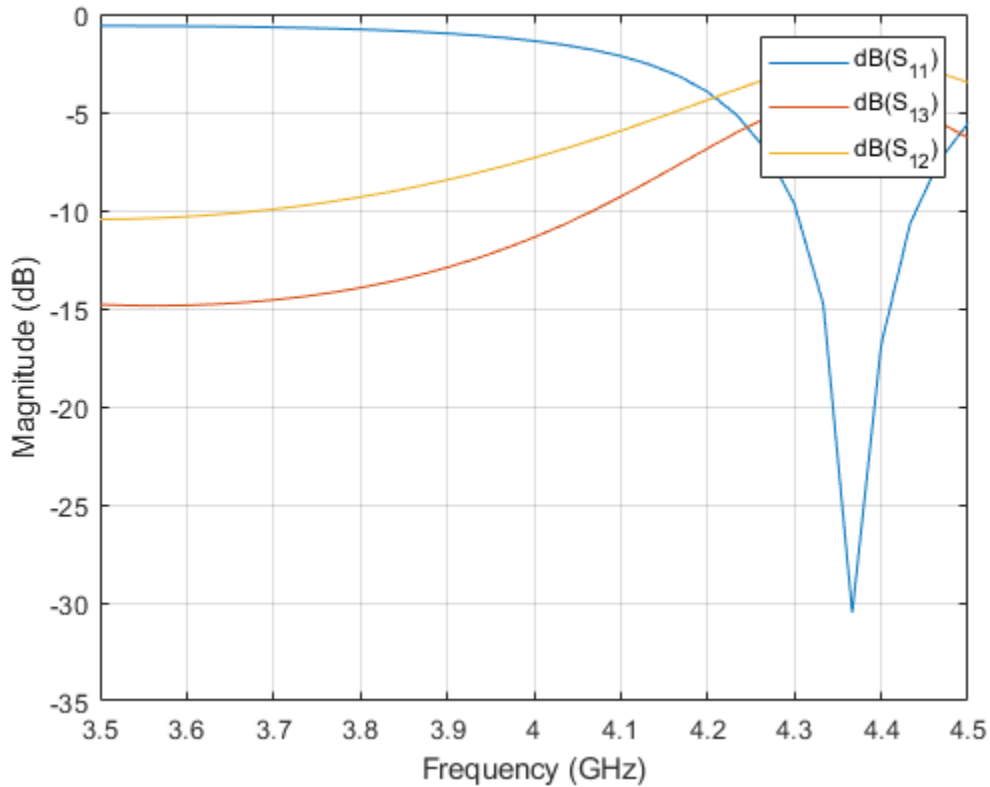


Analyze and plot the S-paramters of this balun.

```
s11 = sparameters(balun, linspace(3.5e9, 4.5e9, 31));
```

```
figure; rfplot(s11, 1, 1);  
hold on; rfplot(s11, 1, 3)  
hold on; rfplot(s11, 1, 2)
```

Click legend labels to toggle the line visibility



Version History

Introduced in R2022a

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

couplerRatrace

traceStep

Create step trace in XY plane

Description

Use the `traceStep` object to create a step trace in the XY plane.

Creation

Syntax

```
trace = traceStep
trace = traceStep(Name=Value)
```

Description

`trace = traceStep` creates a step trace in the XY plane.

`trace = traceStep(Name=Value)` sets "Properties" on page 1-166 using one or more name-value arguments. For example, `traceStep(ReferencePoint=[1 1])` creates a step trace with the reference point [1 1]. Properties not specified retain their default values.

Properties

Name — Name of step trace

'mytraceStep' (default) | character vector | string scalar

Name of the step trace, specified as a character vector or a string scalar.

Example: `trace = traceStep(Name="traceStep")`

Data Types: `char` | `string`

ReferencePoint — X and Y coordinate of step trace

[0 0] (default) | two-element vector

X and Y coordinate of the step trace in meters, specified as a two-element vector of real elements.

Example: `trace = traceStep(ReferencePoint=[1 1])`

Data Types: `double`

Length — Length of each section of step trace

[0.0050 0.0280 0.0260] (default) | vector

Length of each section of the step trace in meters, specified as a vector of positive elements.

Example: `trace = traceStep(Length=[0.0300 0.0200 0.0230])`

Data Types: `double`

Width — Width of each section of step trace`[0.0020 0.0050 0.0090]` (default) | vector

Width of each section of the step trace in meters, specified as a vector of positive elements.

Example: `trace = traceStep(Width=[0.0060 0.0060 0.0080])`

Data Types: double

Symmetry — Symmetry of step trace along X-axis`1` (default) | 0

Symmetry of the step trace along the X-axis, specified as 1 or 0. If set to 1, symmetry is enabled. Set the property to 1 to create a symmetric step trace and to 0 to create a nonsymmetric step trace.

Example: `trace = traceStep(Symmetry=0)`

Data Types: logical

Object Functions

<code>add</code>	Boolean unite operation on two RF PCB shapes
<code>subtract</code>	Boolean subtraction operation on two RF PCB shapes
<code>intersect</code>	Boolean intersection operation on two RF PCB shapes
<code>plus</code>	Shape1 + Shape2 for RF PCB shapes
<code>minus</code>	Shape1 - Shape2 for RF PCB shapes
<code>and</code>	Shape1 & Shape2 for RF PCB shapes
<code>area</code>	Calculate area of RF PCB shape in square meters
<code>rotate</code>	Rotate RF PCB shape about defined axis
<code>rotateX</code>	Rotate RF PCB shape about x-axis
<code>rotateY</code>	Rotate RF PCB shape about y-axis and angle
<code>rotateZ</code>	Rotate RF PCB shape about z-axis
<code>translate</code>	Move RF PCB shape to new location
<code>scale</code>	Change size of RF PCB shape by fixed amount

Examples**Default Step Trace**

Create a step trace with default values.

```
trace = traceStep
```

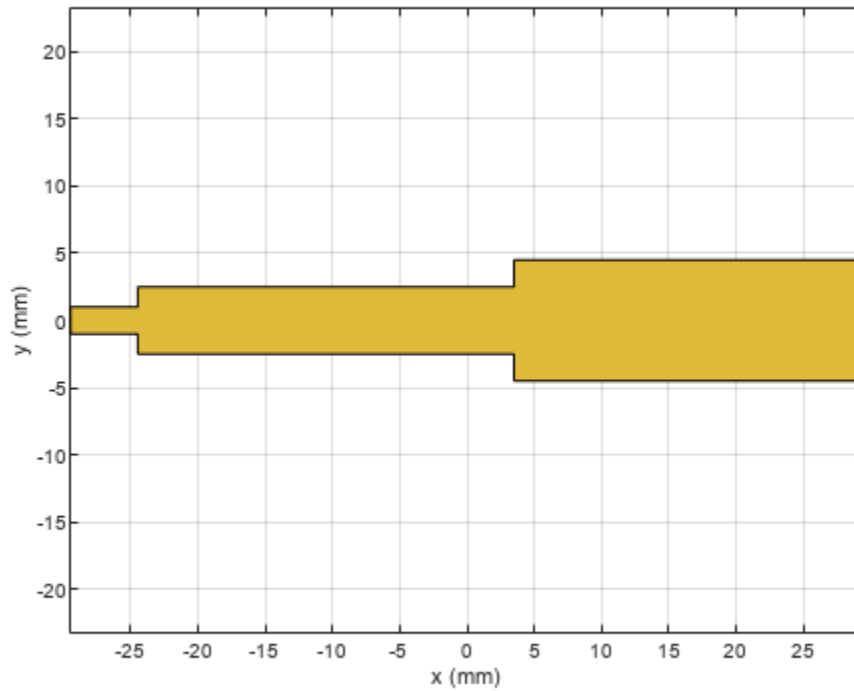
```
trace =
```

```
    traceStep with properties:
```

```
        Name: 'mytraceStep'  
    ReferencePoint: [0 0]  
        Length: [0.0050 0.0280 0.0260]  
        Width: [0.0020 0.0050 0.0090]  
    Symmetry: 1
```

Visualize the step trace.

```
show(trace)
```



Nonsymmetrical Step Trace

Create a nonsymmetrical stepped trace.

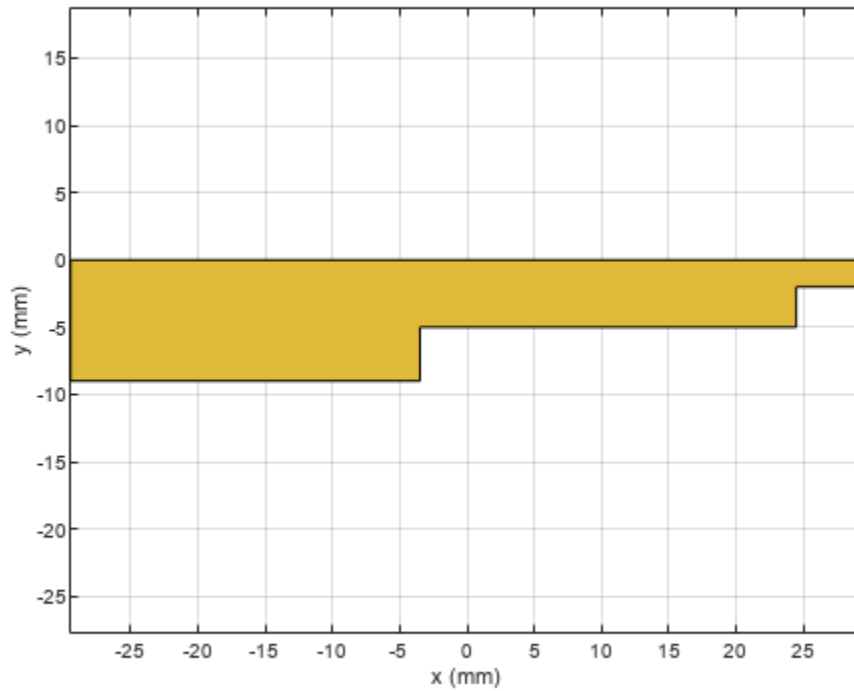
```
trace = traceStep;  
trace.Symmetry = 0;
```

Rotate the step trace by 180 degrees about the Z-axis.

```
trace = rotateZ(trace,180);
```

Visualize the step trace.

```
show(trace)
```



Version History

Introduced in R2022a

See Also

[traceLine](#) | [traceCross](#) | [traceRectangular](#) | [tracePoint](#) | [traceSpiral](#)

traceTapered

Create tapered trace in X-Y plane

Description

Use the `traceTapered` object to create a tapered trace in the X-Y plane.

Creation

Syntax

```
trace = traceTapered
trace = traceTapered(Name=Value)
```

Description

`trace = traceTapered` creates a tapered trace in the X-Y plane.

`trace = traceTapered(Name=Value)` sets "Properties" on page 1-166 using one or more name-value arguments. For example, `traceTapered(ReferencePoint=[1 1])` creates a tapered trace with the reference point [1 1]. Properties not specified retain their default values.

Properties

Name — Name of tapered trace

'mytraceTapered' (default) | character vector | string scalar

Name of the tapered trace, specified as a character vector or a string scalar.

Example: `trace = traceTapered(Name="traceTapered")`

Data Types: `char` | `string`

ReferencePoint — X and Y coordinate of tapered trace

[0 0] (default) | two-element vector

X and Y coordinate of the tapered trace in meters, specified as a two-element vector of real elements.

Example: `trace = traceTapered(ReferencePoint=[1 1])`

Data Types: `double`

CurvatureRate — Curvature rate of tapered trace

50 (default) | real scalar

Curvature rate of the tapered trace in meters, specified as a real scalar. Set the property to 0 to create a linear trace.

Example: `trace = traceStep(CurvatureRate=20)`

Data Types: `double`

InputWidth — Width of tapered trace at input end

0.0050 (default) | positive scalar

Width of the tapered trace at the input end in meters, specified as a positive scalar

Example: `trace = traceTrapered(Width=0.0060)`

Data Types: double

OutputWidth — Width of tapered trace at output end

0.02 (default) | positive scalar

Width of the output side of the taper trace in meters, specified as a positive scalar

Example: `trace = traceTrapered(Width=0.070)`

Data Types: double

Length — Length of tapered trace

0.03 (default) | positive scalar

Length of the tapered trace in meters, specified as a positive scalar.

Example: `trace = traceTrapered(Length=0.040)`

Data Types: double

Symmetry — Symmetry of tapered trace along X-axis

1 (default) | 0

Symmetry of tapered trace along X-axis, specified as 1 or 0. If set to 1, symmetry is enabled.

Example: `trace = traceTapered(Symmetry=0)`

Data Types: logical

Object Functions

add	Boolean unite operation on two RF PCB shapes
subtract	Boolean subtraction operation on two RF PCB shapes
intersect	Boolean intersection operation on two RF PCB shapes
plus	Shape1 + Shape2 for RF PCB shapes
minus	Shape1 - Shape2 for RF PCB shapes
and	Shape1 & Shape2 for RF PCB shapes
area	Calculate area of RF PCB shape in square meters
rotate	Rotate RF PCB shape about defined axis
rotateX	Rotate RF PCB shape about x-axis
rotateY	Rotate RF PCB shape about y-axis and angle
rotateZ	Rotate RF PCB shape about z-axis
translate	Move RF PCB shape to new location
scale	Change size of RF PCB shape by fixed amount

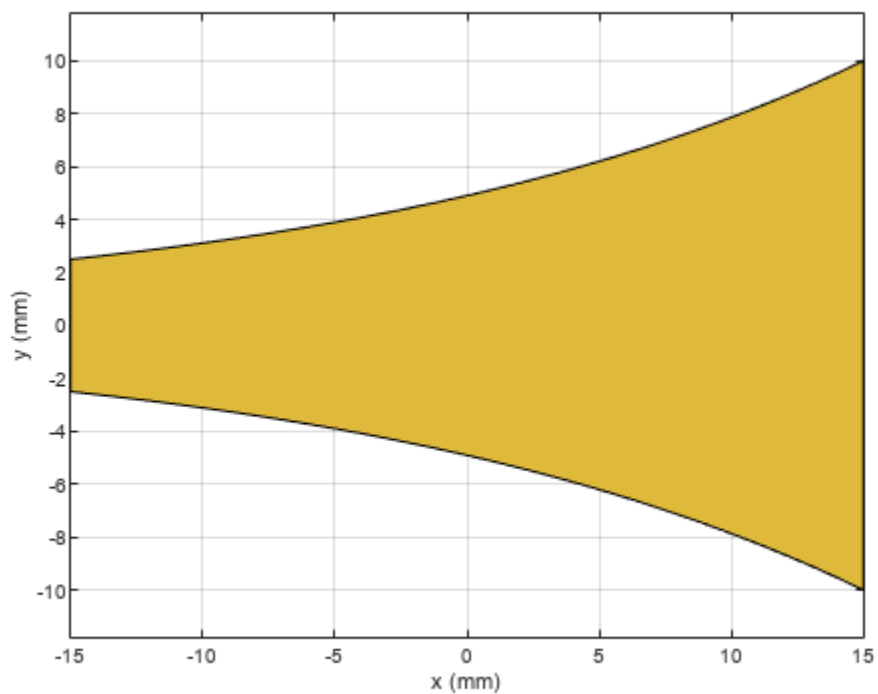
Examples**Default Tapered Trace**

Create a tapered trace with default values.


```
trace = traceTapered
trace =
  traceTapered with properties:
      Name: 'mytraceTapered'
  ReferencePoint: [0 0]
  CurvatureRate: 50
  InputWidth: 0.0050
  OutputWidth: 0.0200
  Length: 0.0300
  Symmetry: 1
```

Visualize the step trace.

```
show(trace)
```



Nonsymmetrical Tapered Trace

Create a nonsymmetrical tapered trace.

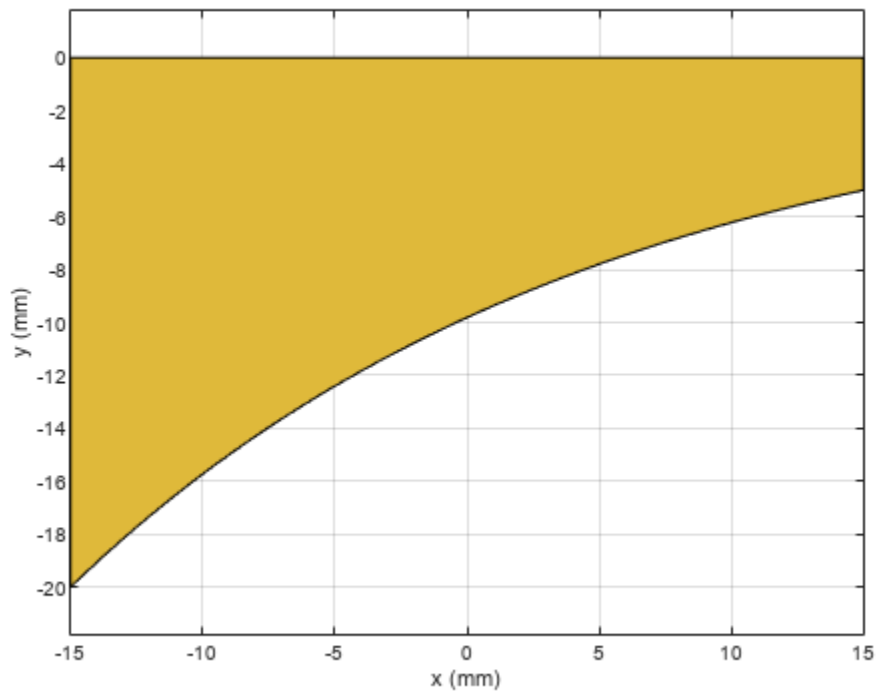
```
trace = traceTapered;
trace.Symmetry = 0;
```

Rotate by 180 degrees about the Z-axis.

```
trace = rotateZ(trace,180);
```

Visualize the step trace.

```
show(trace)
```



Version History

Introduced in R2022a

See Also

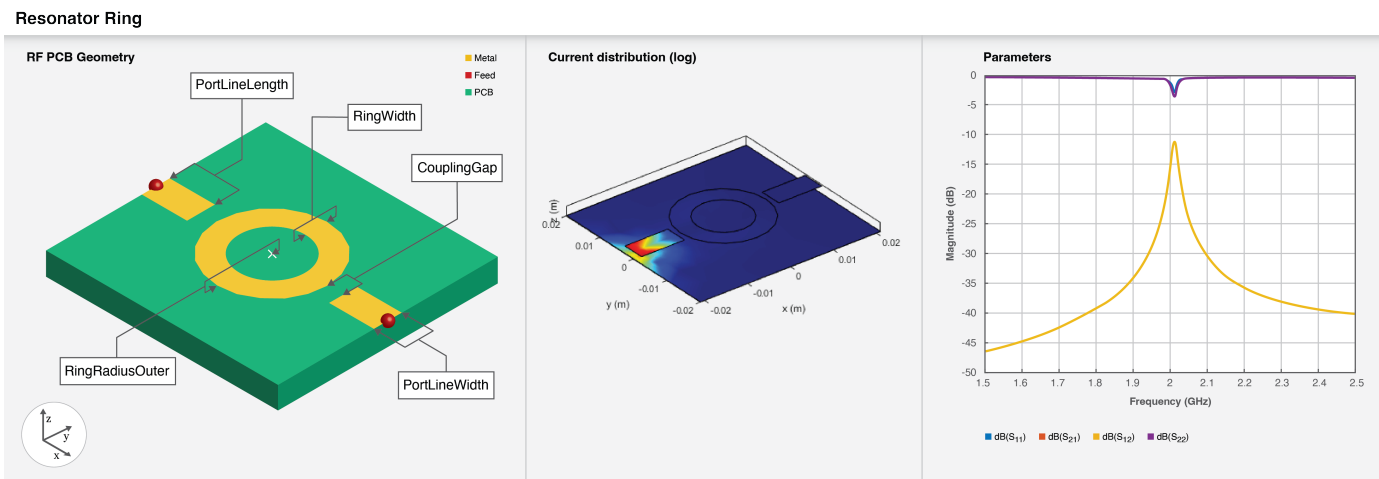
[traceLine](#) | [traceCross](#) | [traceRectangular](#) | [tracePoint](#) | [traceSpiral](#) | [traceStep](#)

resonatorRing

Create gap-coupled ring resonator in microstrip form

Description

Use the `resonatorRing` object to create a gap-coupled ring resonator in the microstrip form. Resonant structures such as rectangular, circular, and ring resonators are used in oscillators and filters.



Creation

Syntax

```
resonator = resonatorRing
resonator = resonatorRing(Name=Value)
```

Description

`resonator = resonatorRing` creates a default gap-coupled ring resonator. The default properties of the ring resonator are for a design frequency of 1 GHz.

`resonator = resonatorRing(Name=Value)` sets “Properties” on page 1-255 using one or more name-value arguments. For example, `resonatorRing(RingWidth=0.045)` creates a ring resonator with a ring width of 0.045 meters. Properties not specified retain their default values

Properties

PortLineLength — Length of port line

0.0100 (default) | positive scalar

Length of the port line in meters, specified as a positive scalar.

Example: `resonator = resonatorRing(PortLineLength=0.045)`

Data Types: double

PortLineWidth — Width of port line

0.0050 (default) | positive scalar

Width of the port line in meters, specified as a positive scalar.

Example: `resonator = resonatorRing(PortLineWidth=0.0061)`

Data Types: double

CouplingGap — Gap between port lines and ring

0.001 (default) | positive scalar

Gap between the port lines and the ring in meters, specified as a positive scalar.

Example: `resonator = resonatorRing(CouplingGap=0.002)`

Data Types: double

RingRadiusOuter — Outer radius of ring

0.0100 (default) | positive scalar

Outer radius of the ring in meters, specified as a positive scalar.

Example: `resonator = resonatorRing(RingRadiusOuter=0.0481)`

Data Types: double

RingWidth — Width of ring

0.0051 (default) | positive scalar

Width of the ring in meters, specified as a positive scalar.

Example: `resonator = resonatorRing(RingWidth=0.0061)`

Data Types: double

Height — Height of resonator from ground plane

0.0016 (default) | positive scalar

Height of the resonator from the ground plane in meters, specified as a positive scalar. In the case of a multilayer substrate, you can use the `Height` property to create a resonator where the two dielectrics interface.

Example: `resonator = resonatorRing(Height=0.020)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0400 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `resonator = resonatorRing(GroundPlaneWidth=0.2241)`

Data Types: double

Substrate – Type of dielectric material

```
dielectric("Teflon") (default) | dielectric object
```

Type of dielectric material used as a substrate, specified as a `dielectric` object. The thickness of the default dielectric material Teflon is 0.0016 m or the same value as the `Height` property.

```
Example: d = dielectric("FR4"); resonator = resonatorRing(Substrate=d)
```

Conductor – Type of metal used in conducting layers

```
metal("PEC") (default) | metal object
```

Type of metal used in the conducting layers, specified as a `metal` object.

```
Example: m = metal("Copper"); resonator = resonatorRing(Conductor=m)
```

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>design</code>	Design ring resonator around specified frequency
<code>feedCurrent</code>	Calculate current at feed port
<code>getZ0</code>	Calculate characteristic impedance of transmission line
<code>layout</code>	Plot all metal layers and board shape
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>shapes</code>	Extract all metal layer shapes of PCB component
<code>show</code>	Display PCB component structure or PCB shape
<code>sparameters</code>	Calculate S-parameters for RF PCB objects

Examples**Default Ring Resonator**

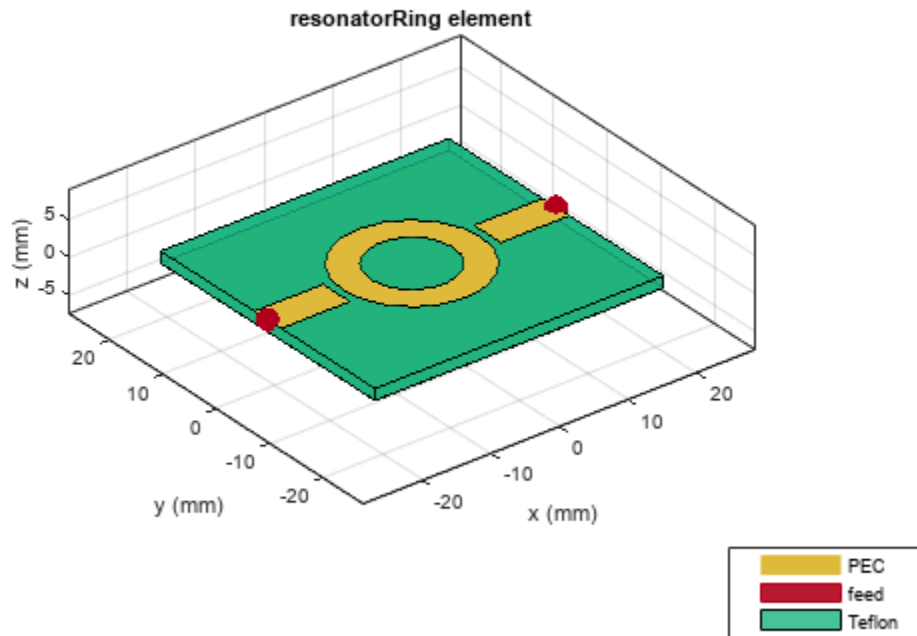
Create a default ring resonator.

```
resonator = resonatorRing
```

```
resonator =
  resonatorRing with properties:
      PortLineLength: 0.0100
      PortLineWidth: 0.0050
      CouplingGap: 1.0000e-03
      RingRadiusOuter: 0.0100
      RingWidth: 0.0040
      Height: 0.0016
      GroundPlaneWidth: 0.0400
      Substrate: [1x1 dielectric]
      Conductor: [1x1 metal]
```

Visualize the resonator.

```
show(resonator)
```



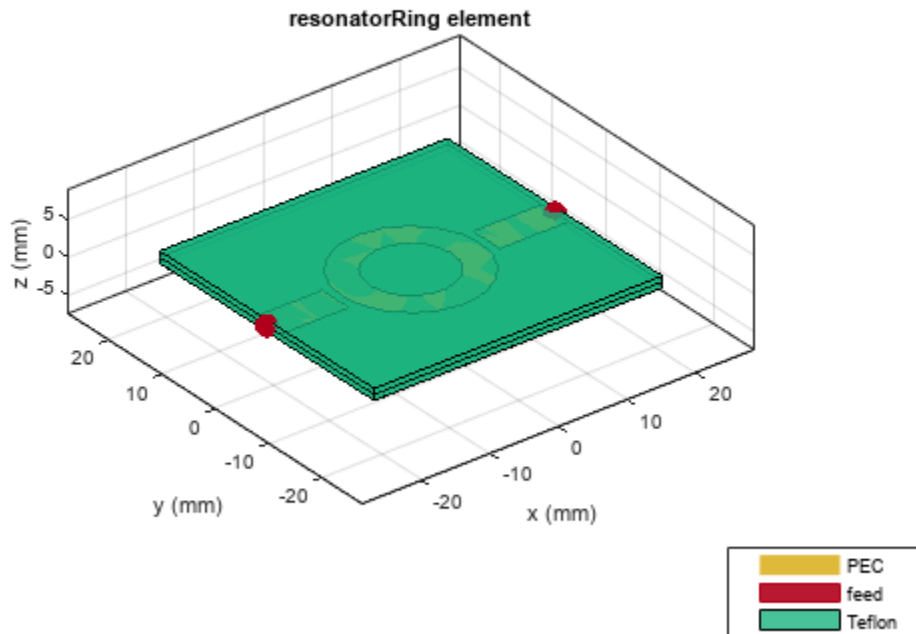
Multilayer Dielectric Ring Resonator

Create a ring resonator at the interface of a multilayer dielectric.

```
resonator = resonatorRing;  
resonator.Substrate = dielectric(Name={'Teflon', 'Teflon'}, EpsilonR=[2.1 2.1], ...  
    LossTangent=[0 0], Thickness=[0.8e-3 0.8e-3]);  
resonator.Height = 0.8e-3;
```

Visualize the ring resonator.

```
show(resonator)
```



Version History

Introduced in R2022b

References

- [1] Pozar, David.M. *Microwave Engineering* Singapore; JohnWiley and Sons. Inc, 2012.
- [2] Bogner, Andreas, Carsten Steiner, Stefanie Walter, Jaroslaw Kita, Gunter Hagen, and Ralf Moos. *Planar Microstrip ring resonators for Microwave-Based Gas Sensing: Design Aspects and Initial Transducers for Humidity and Ammonia Sensing*.
- [3] Bernard, P.A., and J.M. Gautray. "Measurement of Dielectric Constant Using a Microstrip Ring Resonator." *IEEE Transactions on Microwave Theory and Techniques* 39, no. 3 (March 1991): 592-95.

See Also

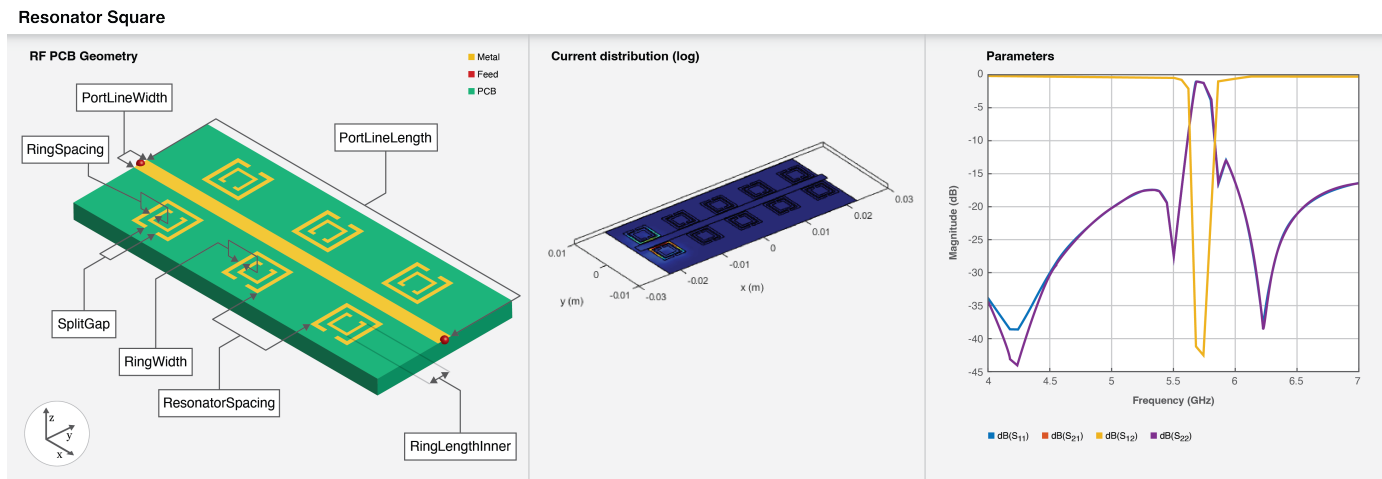
resonatorSplitRingSquare

resonatorSplitRingSquare

Create microstrip line loaded with square split-ring resonator

Description

Use the `resonatorSplitRingSquare` object to create a microstrip line loaded with a square split-ring resonator.



Creation

Syntax

```
resonator = resonatorSplitRingSquare
resonator = resonatorSplitRingSquare(Name=Value)
```

Description

`resonator = resonatorSplitRingSquare` creates a microstrip line loaded with a square split-ring resonator on the X-Y plane. The default properties are for a design frequency of 5.7 GHz.

`resonator = resonatorSplitRingSquare(Name=Value)` sets “Properties” on page 1-260 using one or more name-value arguments. For example, `resonatorSplitRingSquare(RingWidth=0.045)` creates a square split-ring resonator with a ring width of 0.045 meters. Properties not specified retain their default values

Properties

RingLengthInner — Length of inner ring

0.0036 (default) | positive scalar

Length of the inner ring of the ring resonator in meters, specified as a positive scalar.

Example: resonator = resonatorSplitRingSquare(RingLengthInner=0.045)

Data Types: double

RingWidth — Width of square ring

5.0000e-04 (default) | positive scalar

Width of the square ring in meters, specified as a positive scalar.

Example: resonator = resonatorSplitRingSquare(RingWidth=0.0009)

Data Types: double

RingSpacing — Spacing between inner and outer square ring

3.0000e-04 (default) | positive scalar

Spacing between the inner and outer square ring in meters, specified as a positive scalar.

Example: resonator = resonatorSplitRingSquare(RingSpacing=0.0008)

Data Types: double

SplitGap — Split gap on square ring

5.0000e-04 (default) | positive scalar

Split gap on the square ring in meters, specified as a positive scalar.

Example: resonator = resonatorSplitRingSquare(SplitGap=0.0008)

Data Types: double

CouplingGap — Gap between resonator and microstrip line

2.5000e-04 (default) | positive scalar

Gap between the resonator and the microstrip line in meters, specified as a positive scalar.

Example: resonator = resonatorSplitRingSquare(CouplingGap=0.00082)

Data Types: double

NumResonator — Number of resonators on one side of transmission line

5 (default) | positive scalar in the range [1,7]

Number of resonators on one side of the transmission line, specified as a positive scalar in the range [1,7]. The object supports a minimum of one resonator and a maximum of seven resonators.

Example: resonator = resonatorSplitRingSquare(NumResonator=4)

Data Types: double

ResonatorSpacing — Spacing between resonators

0.004 (default) | positive scalar

Spacing between the resonators when number of resonators is more than one in meters, specified as a positive scalar.

Example: resonator = resonatorSplitRingSquare(NumResonator=4)

Data Types: double

PortLineLength — Length of microstrip line`0.0450` (default) | positive scalar

Length of the microstrip line in meters, specified as a positive scalar.

Example: `resonator = resonatorSplitRingSquare(PortLineLength=0.055)`

Data Types: double

PortLineWidth — Width of microstrip line`0.0019` (default) | positive scalar

Width of the microstrip line in meters, specified as a positive scalar.

Example: `resonator = resonatorSplitRingSquare(PortLineWidth=0.0061)`

Data Types: double

Height — Height of resonator from ground plane`8.1300e-04` (default) | positive scalar

Height of the resonator from the ground plane in meters, specified as a positive scalar.

Example: `resonator = resonatorSplitRingSquare(Height=0.020)`

Data Types: double

GroundPlaneWidth — Width of ground plane`0.0200` (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `resonator = resonatorSplitRingSquare(GroundPlaneWidth=0.2241)`

Data Types: double

Substrate — Type of dielectric material`dielectric("FR4")` (default) | dielectric object

Type of dielectric material to use as a substrate, specified as a dielectric object. The thickness of the default dielectric material has the following parameters:

(Name={'FR4'}, EpsilonR=3.38, LossTangent=0.0002, Thickness=0.813e-3).

Example: `d = dielectric("FR4"); resonator = resonatorSplitRingSquare(Substrate=d)`

Conductor — Type of metal in conducting layers`metal("Copper")` (default) | metal object

Type of metal to use in the conducting layers, specified as a metal object.

Example: `m = metal("PEC"); resonator = resonatorSplitRingSquare(Conductor=m)`

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>design</code>	Design hairpin filter around specified frequency
<code>feedCurrent</code>	Calculate current at feed port

getZ0	Calculate characteristic impedance of transmission line
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples

Default Square Split-Ring Resonator

Create a default square split-ring resonator.

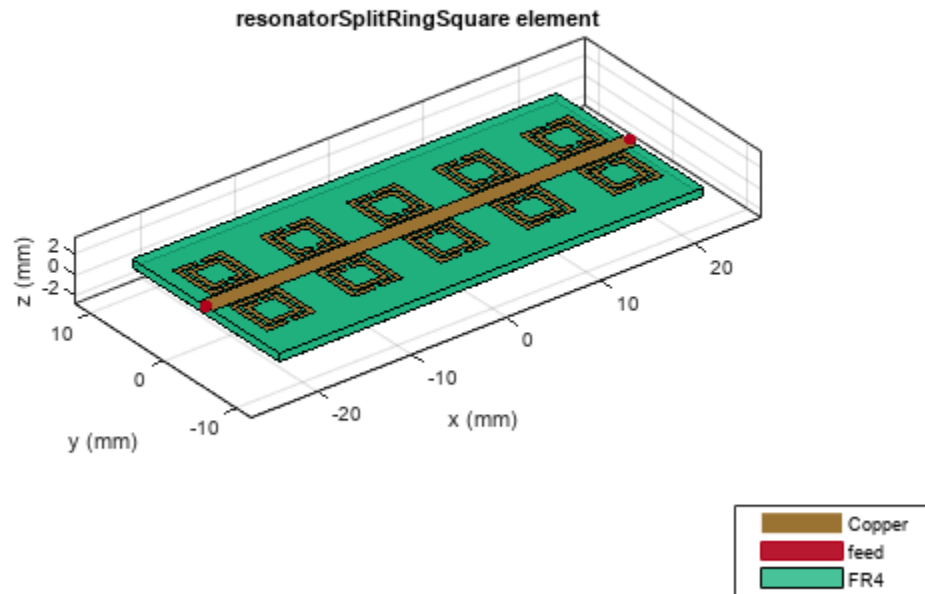
```
resonator = resonatorSplitRingSquare
```

```
resonator =  
    resonatorSplitRingSquare with properties:
```

```
    RingLengthInner: 0.0036  
        RingWidth: 5.0000e-04  
        RingSpacing: 3.0000e-04  
            SplitGap: 5.0000e-04  
        CouplingGap: 2.5000e-04  
        NumResonator: 5  
    ResonatorSpacing: 0.0040  
        PortLineLength: 0.0450  
        PortLineWidth: 0.0019  
            Height: 8.1300e-04  
    GroundPlaneWidth: 0.0200  
        Substrate: [1x1 dielectric]  
        Conductor: [1x1 metal]
```

View the resonator.

```
show(resonator)
```



Square Split-Ring Resonator with Different Dielectrics

Create a split-ring square with one resonator.

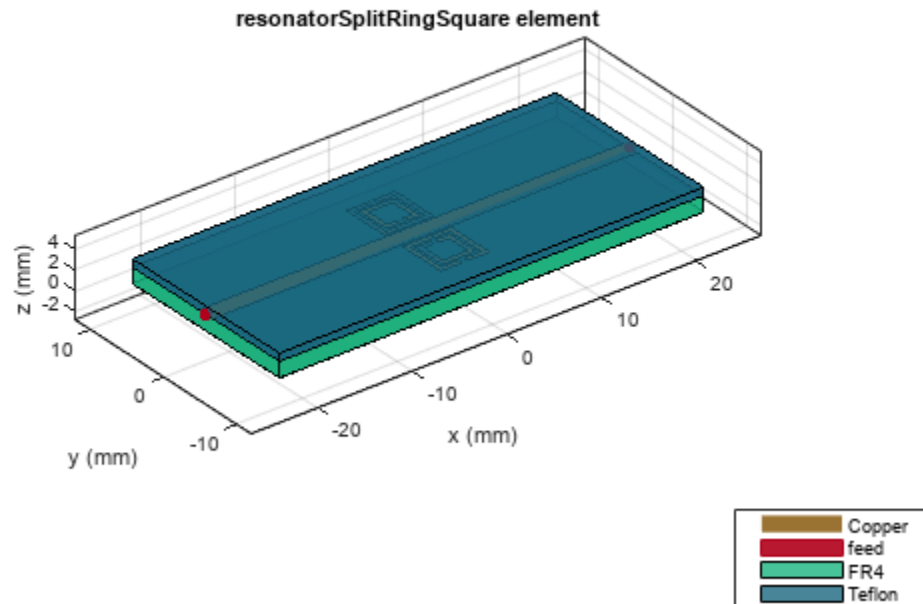
```
resonator = resonatorSplitRingSquare(NumResonator=1);
resonator.Height = 0.0016;
```

Create a dielectric object with two dielectrics. Use this substrate in the resonator.

```
sub = dielectric('FR4','Teflon');
sub.Thickness = [0.0016 0.0008];
resonator.Substrate = sub;
```

View the resonator.

```
figure;
show(resonator);
```



Version History

Introduced in R2022b

References

- [1] Pozar, David.M. *Microwave Engineering* Singapore; JohnWiley and Sons. Inc, 2012.
- [2] Mahyuddin, Muzlifah and Nur Farah Syazwani Ab. Kadir. *Design of a 5.8 GHz Bandstop Filter Using Split Ring Resonator Array*
- [3] Mahyuddin, Muzlifah and Nur Farah Syazwani Ab. Kadir. "A 10 GHz Low Phase Noise Split-Ring Resonator Oscillator." *International Journal of Information and Electronics Engineering*, 2013.

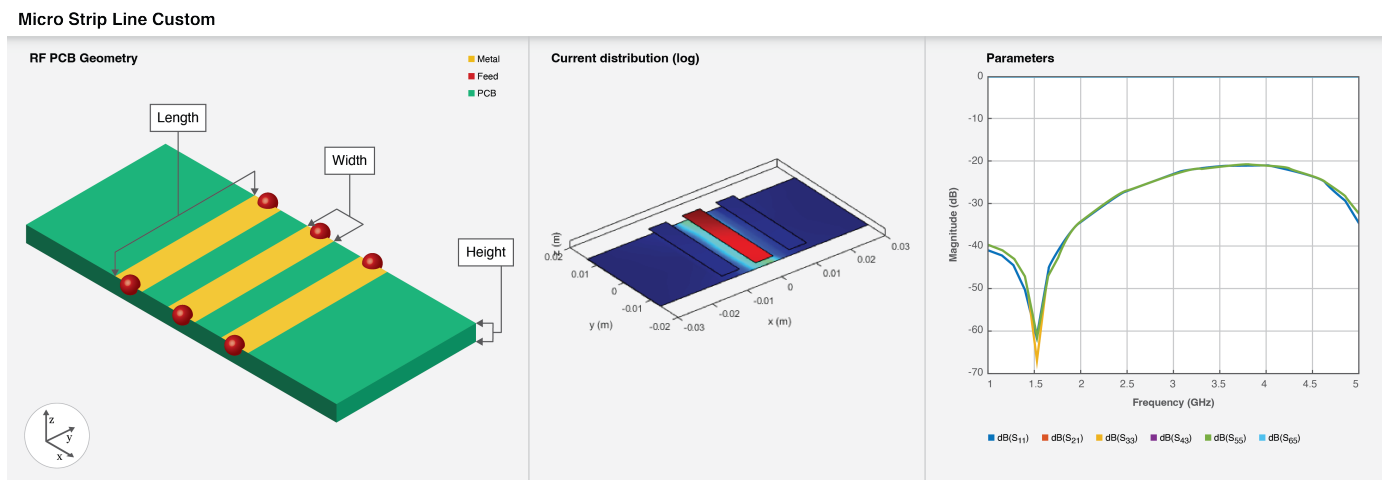
See Also

microstripLineCustom

Create coupled form of single or differential microstrip transmission line

Description

Use the `microstripLineCustom` object to create a coupled form of single or differential microstrip transmission line. A microstrip line is a transmission line that is a basic building block for most RF planar microwave devices. You can use this transmission line to connect two PCB components or to create components such as filters, couplers, and feeding elements of several types of antennas.



A few applications of microstrip transmission lines are:

- Creating matching feed and coupling networks
- Transmitting power from one component to another
- Feeding planar antennas and coupling structures
- Creating varying inductances or capacitances using open- or short ended- transmission lines

Creation

Syntax

```
microstrip = microstripLineCustom
microstrip = microstripLineCustom(Name=Value)
```

Description

`microstrip = microstripLineCustom` creates a coupled form of single or differential microstrip transmission line. The default properties are for a design frequency of 2.5 GHz.

`microstrip = microstripLineCustom(Name=Value)` sets properties using one or more name value pair arguments. For example, `microstrip =`

`microstripLineCustom(TraceSpacing=0.0300)` creates a custom differential microstrip transmission line with a trace spacing of 0.0300 meters. Properties not specified retain their default values.

Properties

TraceType — Types of traces

'Single' (default) | 'Differential'

Types of traces, specified as 'Single' or 'Differential'.

Example: `microstrip = microstripLineCustom(TraceType='Differential')`

Data Types: string | char

TraceLength — Length of trace

0.0271 (default) | positive scalar

Length of the trace in meters, specified as a positive scalar.

Example: `microstrip = microstripLineCustom(TraceLength=0.0371)`

Data Types: double

TraceWidth — Width of trace

0.0051 (default) | positive scalar

Width of the trace in meters, specified as a positive scalar.

Example: `microstrip = microstripLineCustom(TraceWidth=0.0041)`

Data Types: double

TraceSpacing — Spacing between two traces of differential pair

0.0046 (default) | positive scalar

Spacing between the two traces of a differential pair in meters, specified as a positive scalar. Set the `TraceType` property to 'Differential' to enable this property.

Example: `microstrip = microstripLineCustom(TraceSpacing=0.0041)`

Data Types: double

TraceOffsetX — Position to offset reference trace along X-axis

0 (default) | scalar

Position to offset reference trace along the X-axis, specified as a positive scalar.

Example: `microstrip = microstripLineCustom(TraceOffsetX=0.5)`

Data Types: double

Height — Height from custom microstrip line to ground plane

0.0016 (default) | positive scalar

Height from the custom microstrip line to the ground plane in meters, specified as a positive scalar.

In the case of a multilayer substrate, you can use the height property to create a microstrip line at the interface of the two dielectrics.

Example: `microstrip = microstripLineCustom(Height=0.0026)`

Data Types: double

GroundPlaneLength — Length of ground plane

`0.0600` (default) | positive scalar

Length of the ground plane in meters, specified as a positive scalar.

Example: `microstrip = microstripLineCustom(GroundPlaneLength=0.0500)`

Data Types: double

LeftCoupledTraceGap — Gap between left coupled traces

`0.0046` (default) | scalar | vector

Gap between the left coupled traces in meters, specified as a scalar or vector. Set this property to `0`, if there are no left coupled traces.

Example: `microstrip = microstripLineCustom(LeftCoupledTraceGap=0)`

Data Types: double

RightCoupledTraceGap — Gap between right coupled traces

`0.0046` (default) | scalar | vector

Gap between the right coupled traces in meters, specified as a scalar or vector. Set this property to `0`, if there are no right coupled traces.

Example: `microstrip = microstripLineCustom(RightCoupledTraceGap=0)`

Data Types: double

Substrate — Type of dielectric material

`dielectric("Teflon")` (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object. For more information, see dielectric. The thickness of the default dielectric material Teflon is `0.0016` m or the same as the height property.

Example: `d = dielectric('FR4'); microstrip = microstripLineCustom(Substrate=d)`

Conductor — Type of metal used in conducting layers

`metal("PEC")` (default) | metal object

Type of metal used in conducting layers, specified as a metal object. For more information see metal.

Example: `m = metal('PEC'); microstrip = microstripLineCustom(Conductor=m)`

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>design</code>	Design microstrip transmission line around specified frequency
<code>feedCurrent</code>	Calculate current at feed port
<code>getZ0</code>	Calculate characteristic impedance of transmission line
<code>layout</code>	Plot all metal layers and board shape

mesh Change and view mesh properties of metal or dielectric in PCB component
 shapes Extract all metal layer shapes of PCB component
 show Display PCB component structure or PCB shape
 sparameters Calculate S-parameters for RF PCB objects

Examples

Differential Right Coupled Microstrip Line

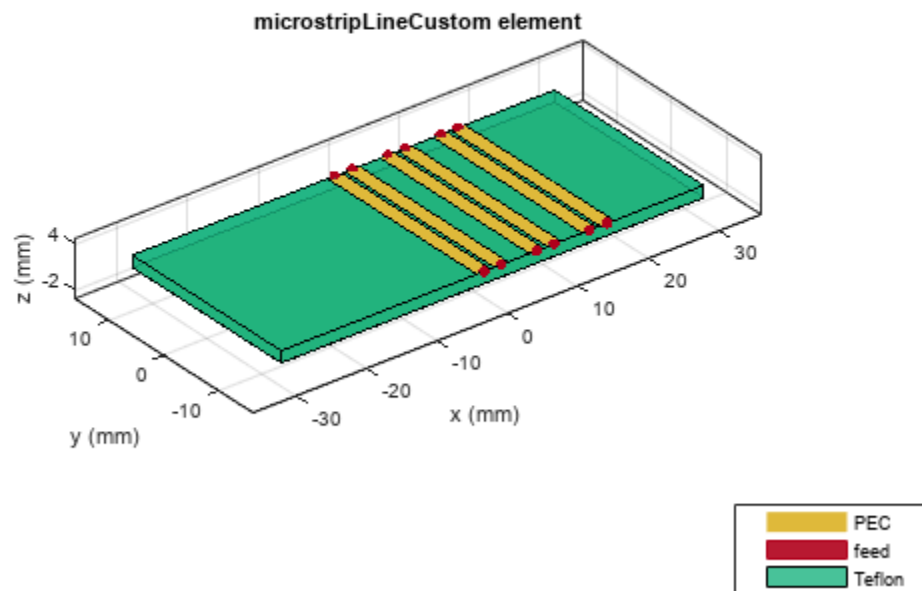
Create a coupled form of differential microstrip line with two pairs of right coupled lines.

```

microstrip = microstripLineCustom(TraceType='Differential',TraceWidth=...
    0.002,TraceSpacing=0.0005,RightCoupledTraceGap=[0.003,0.003],...
    LeftCoupledTraceGap=0);
  
```

View the custom transmission line.

```
show(microstrip)
```



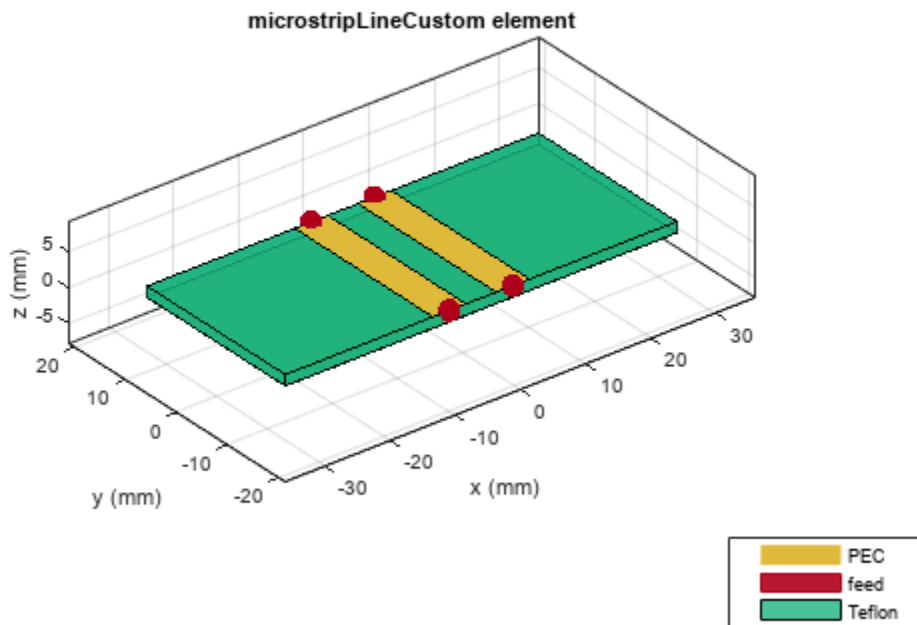
Differential Microstrip Line with No Left or Right Traces

Create a differential microstrip transmission line with no left or right traces

```
microstrip = microstripLineCustom(TraceType='Differential',TraceSpacing=...  
    0.0046,LeftCoupledTraceGap=0,RightCoupledTraceGap=0);
```

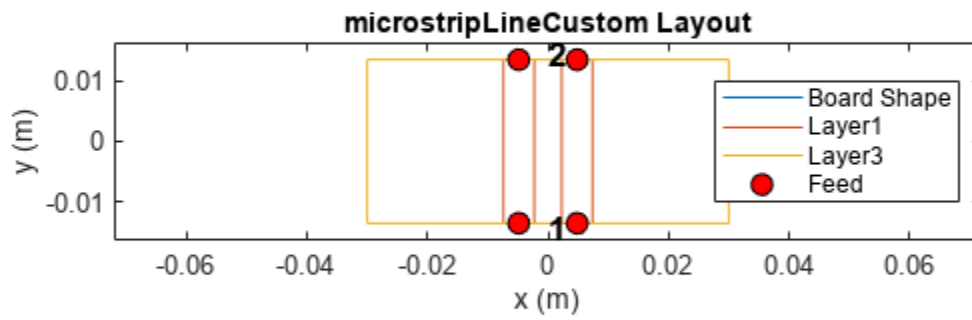
View the transmission line.

```
show(microstrip)
```



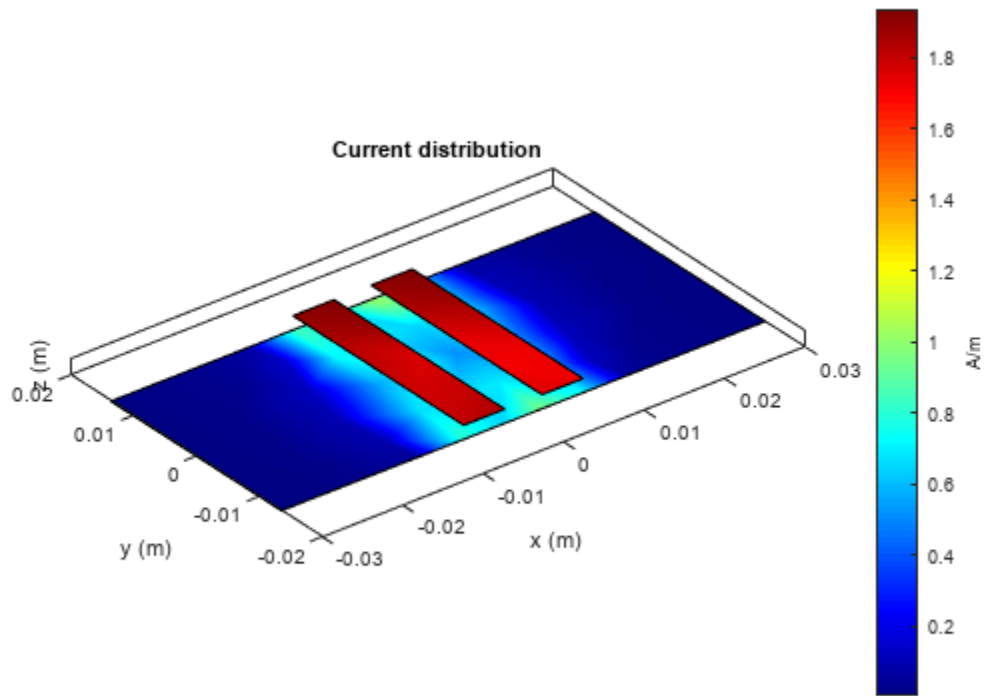
Set the voltage on the transmission line ports. View the layout.

```
v = voltagePort(4);  
v.FeedVoltage = [1 0 1 0];  
v.FeedPhase = [0 0 180 0];  
figure;  
layout(microstrip)
```



Analyze the current at 2.5 GHz.

```
figure;  
current(microstrip,2.5e9,Excitation=v)
```



Version History

Introduced in R2022b

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

`coplanarWaveguide` | `coupledMicrostripline`

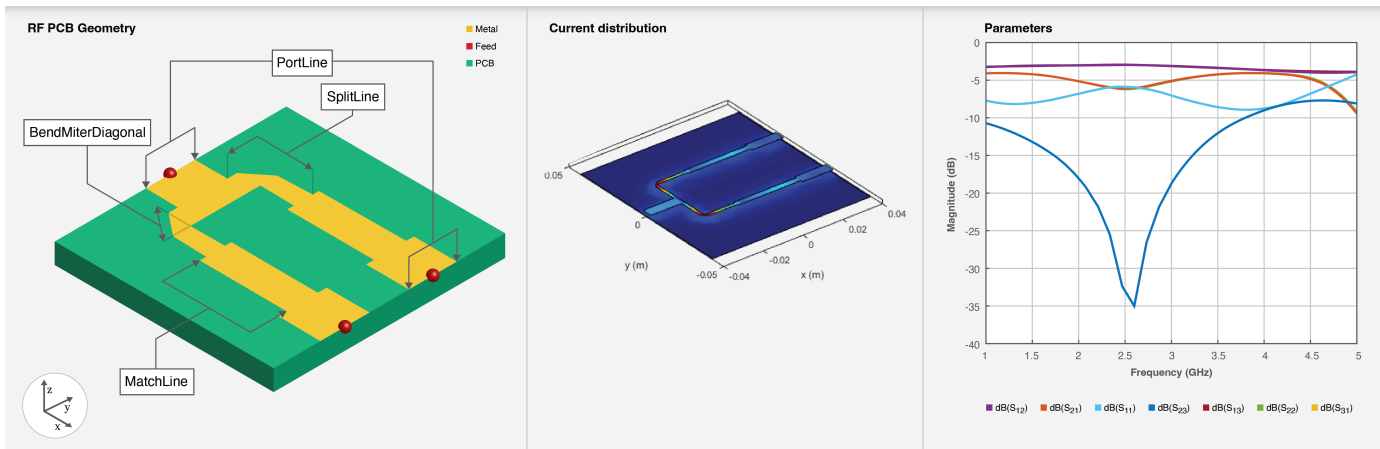
splitterTee

Create T-junction power splitter

Description

Use the `splitterTee` object to create a T-junction power splitter. The T-junction is a lossless and reciprocal three-port divider which divides the input equally between the two output ports.

Splitter Tee



To analyze the behavioral model for the T-junction power splitter, set the Behavioral property in parameters to true or 1.

Creation

Syntax

```
splitter = splitterTee
splitter = splitterTee(Name=Value)
```

Description

`splitter = splitterTee` creates a T-junction power splitter with default properties. The default properties are for a design frequency of 1.8 GHz.

`splitter = splitterTee(Name=Value)` sets “Properties” on page 1-274 using one or more name-value arguments. For example, `splitterTee(PortLineLength=0.0155)` creates a T-junction splitter with a port line length of 0.0155 meters. Properties not specified retain their default values.

Properties

Shape — Shape of T-junction splitter

'RectangularMitered' (default) | 'RectangularCurved' | 'Circular'

Shape of the T-junction splitter, specified as 'RectangularMitered', 'RectangularCurved', or 'Circular'.

Example: `splitter = splitterTee(Shape='Circular')`

Data Types: char

PortLineLength — Length of port line

0.0155 (default) | positive scalar

Length of the port line in meters, specified as a positive scalar.

Example: `splitter = splitterTee(PortLineLength=0.0144)`

Data Types: double

PortLineWidth — Width of port line

0.0051 (default) | positive scalar

Width of the port line in meters, specified as a positive scalar.

Example: `splitter = splitterTee(PortLineWidth=0.0041)`

Data Types: double

SplitLineLength — Length of split line

0.0320 (default) | positive scalar

Length of the split line in meters, specified as a positive scalar. Generally the length is $\lambda/4$ for a Tee splitter.

Example: `splitter = splitterTee(SplitLineLength=0.0420)`

Data Types: double

SplitLineWidth — Width of split line

0.0015 (default) | positive scalar

Width of the split line in meters, specified as a positive scalar.

Example: `splitter = splitterTee(SplitLineWidth=0.0025)`

Data Types: double

BendMiterDiagonal — Length of miter diagonal at bend

0.0035 (default) | positive scalar

Length of the miter diagonal at the bend in meters, specified as a positive scalar.

Example: `splitter = splitterTee(BendMiterDiagonal=0.0045)`

Data Types: double

BendCurveRadius — Radius of curve at bend

0.0020 (default) | positive scalar

Radius of the curve at the bend in meters, specified as a positive scalar.

Example: `splitter = splitterTee(BendCurveRadius=0.0030)`

Dependencies

To enable this property, set the Shape property to 'RectangularCurved'.

Data Types: double

MatchLineLength — Length of match line

0.0315 (default) | positive scalar

Length of the match line in meters, specified as a positive scalar.

Example: `splitter = splitterTee(MatchLineLength=0.0415)`

Data Types: double

MatchLineWidth — Width of match line

0.0029 (default) | positive scalar

Width of the match line in meters, specified as a positive scalar.

Example: `splitter = splitterTee(MatchLineWidth=0.0039)`

Data Types: double

PortSpacing — Spacing between output ports

0.0320 (default) | positive scalar

Spacing between the output ports in meters, specified as a positive scalar.

Example: `splitter = splitterTee(PortSpacing=0.0420)`

Data Types: double

Height — Height of T-junction power splitter from ground plane

0.0016 (default) | positive scalar

Height of the T-junction power splitter from the ground plane in meters, specified as a positive scalar.

In the case of a multilayer substrate, you can use the Height property to create a T-junction power splitter where the two dielectrics interface.

Example: `splitter = splitterTee(Height=0.0076)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0847 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `splitter = splitterTee(GroundPlaneWidth=0.098)`

Example: double

Substrate — Type of dielectric material

dielectric object

Type of dielectric material used as a substrate, specified as a `dielectric` object. The dielectric material in a `splitterTee` object with default properties in Teflon.

```
Example: d = dielectric("FR4"); splitter = splitterTee(Substrate=d)
```

Data Types: `string` | `char`

Conductor — Type of metal used in conducting layers

`metal` object

Type of metal used in the conducting layers, specified as a `metal` object. The metal in a `splitterTee` object with default properties in PEC.

```
Example: m = metal("Copper"); splitter = splitterTee(Conductor=m)
```

Data Types: `string` | `char`

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>design</code>	Design T-junction power splitter around specified frequency
<code>feedCurrent</code>	Calculate current at feed port
<code>layout</code>	Plot all metal layers and board shape
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>shapes</code>	Extract all metal layer shapes of PCB component
<code>show</code>	Display PCB component structure or PCB shape
<code>sparameters</code>	Calculate S-parameters for RF PCB objects

Examples

Default Splitter Tee

Create a default splitter tee.

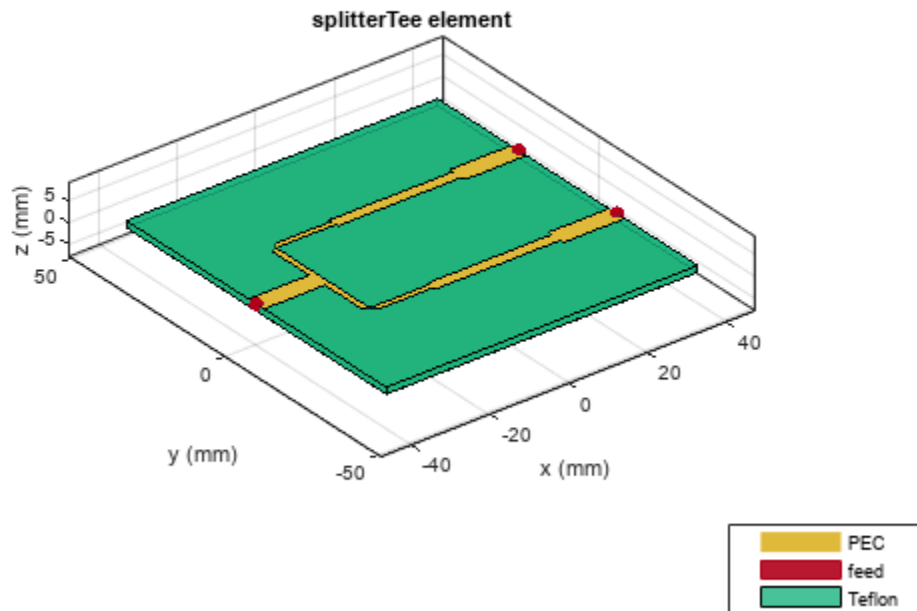
```
splitter = splitterTee
```

```
splitter =  
  splitterTee with properties:
```

```
          Shape: 'RectangularMitered'  
    PortLineLength: 0.0155  
    PortLineWidth: 0.0051  
    SplitLineLength: 0.0320  
    SplitLineWidth: 0.0015  
    BendMiterDiagonal: 0.0035  
    MatchLineLength: 0.0315  
    MatchLineWidth: 0.0029  
    PortSpacing: 0.0320  
    Height: 0.0016  
    GroundPlaneWidth: 0.0847  
    Substrate: [1x1 dielectric]  
    Conductor: [1x1 metal]
```

Visualize the splitter tee.


```
show(splitter)
```



S-Parameters of Curved Rectangular Splitter Tee

Create a curved rectangular splitter tee.

```
splitter = splitterTee(Shape='RectangularCurved')
```

```
splitter =
```

```
splitterTee with properties:
```

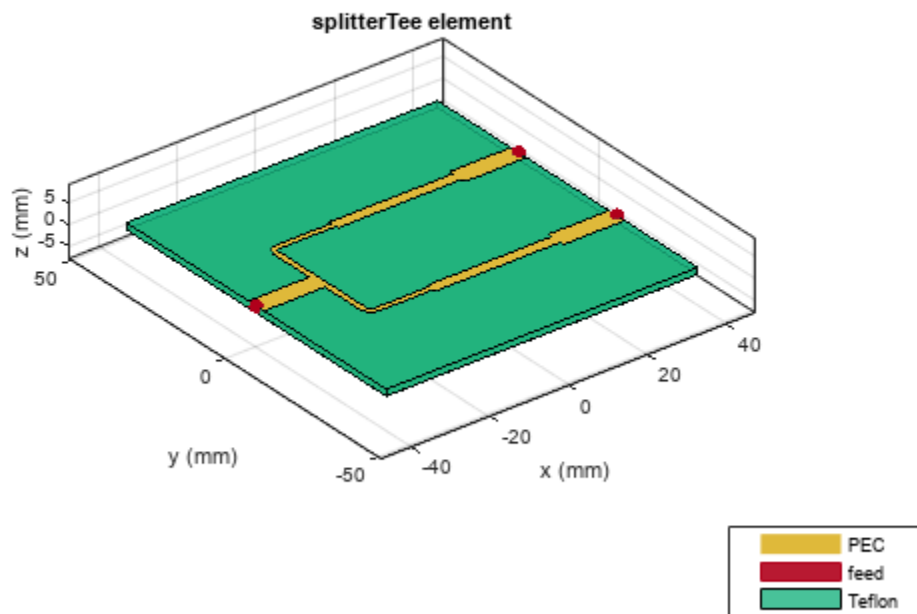
```

    Shape: 'RectangularCurved'
    PortLineLength: 0.0155
    PortLineWidth: 0.0051
    SplitLineLength: 0.0320
    SplitLineWidth: 0.0015
    BendCurveRadius: 0.0020
    MatchLineLength: 0.0315
    MatchLineWidth: 0.0029
    PortSpacing: 0.0320
    Height: 0.0016
    GroundPlaneWidth: 0.0847
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]

```

Visualize the splitter tee.

```
show(splitter)
```



Calculate and plot the s-parameters of the tee at 3 GHz.

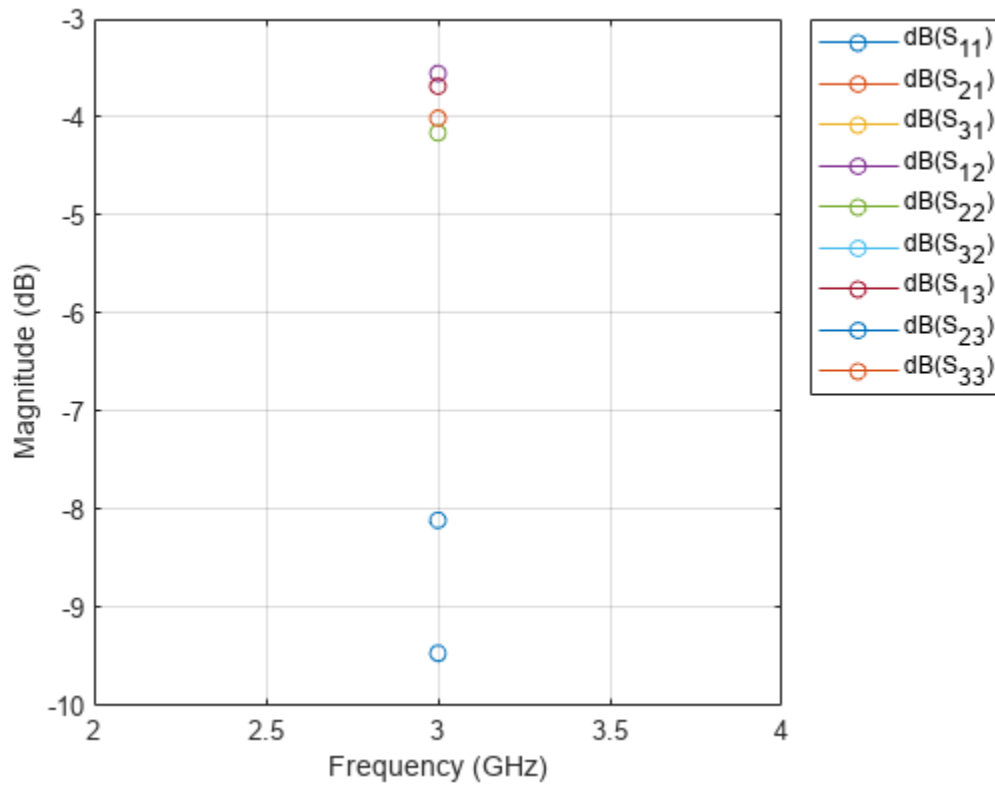
```
spar = sparameters(splitter,3e9)
```

```
spar =  
  sparameters: S-parameters object
```

```
  NumPorts: 3  
  Frequencies: 3.0000e+09  
  Parameters: [3x3 double]  
  Impedance: 50
```

`rfparam(obj,i,j)` returns S-parameter S_{ij}

```
rfplot(spar)
```



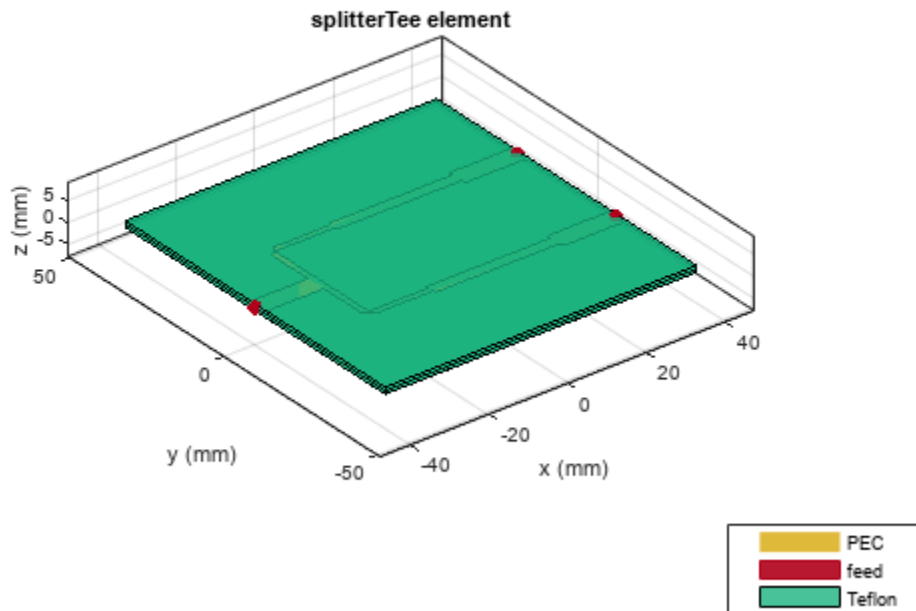
Multilayer Dielectric Splitter Tee

Create a splitter tee at the interface of a multilayer dielectric.

```
splitter = splitterTee;
splitter.Substrate = dielectric(Name={'Teflon','Teflon'},EpsilonR=[2.1 2.1], ...
    LossTangent=[0 0],Thickness=[0.8e-3 0.8e-3]);
splitter.Height = 0.8e-3;
```

Visualize the splitter tee.

```
show(splitter);
```



Version History

Introduced in R2022b

Behavioral Analysis for T-Junction Power Splitter

Use the `sparameters` function and the `pcbElement` object to perform the behavioral analysis of a T-junction power splitter.

References

- [1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.
- [2] Kumari, Chanchala, and Neela Chatteraj. "Design of an Elementary Microstrip Power Splitter for Antenna Array." In 2021 National Conference on Communications (NCC), 1-5. Kanpur, India: IEEE, 2021. <https://doi.org/10.1109/NCC52529.2021.9530097>.

See Also

`wilkinsonSplitter` | `wilkinsonSplitterUnequal` | `wilkinsonSplitterWideband`

Topics

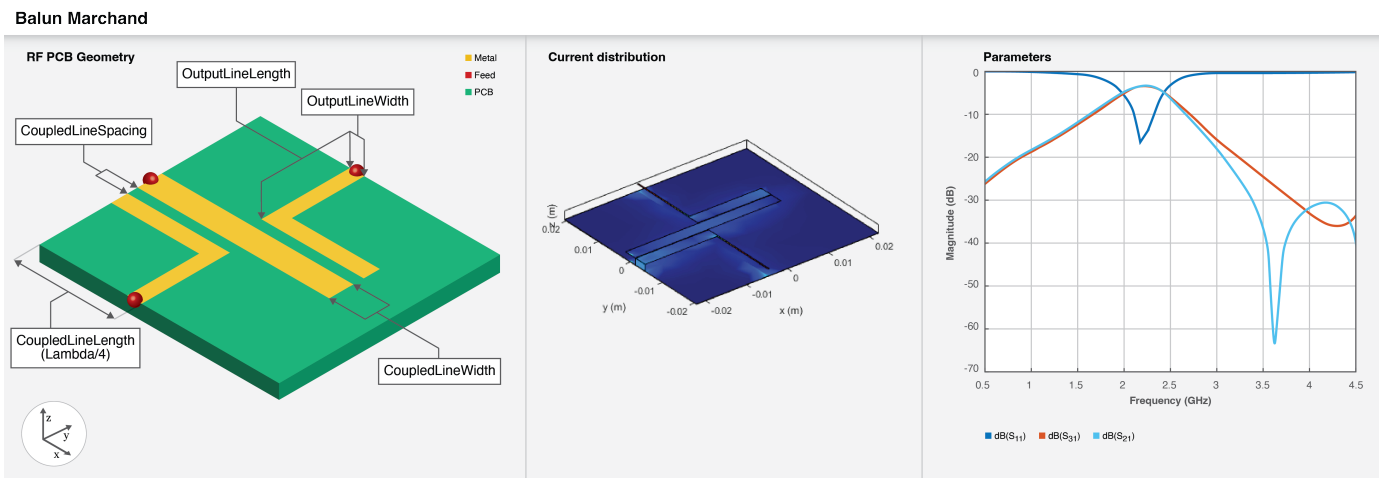
"Behavioral Models"

balunMarchand

Create Marchand balun in microstrip form

Description

Use the `balunMarchand` object to create a Marchand balun in the microstrip form with an unbalanced and balanced output. There is a 180 degrees phase difference between the input and the output ports.



Baluns are used to connect an unbalanced source line to a balanced load or vice-versa enabling you to create a matching transition between balanced and unbalanced transmission lines. A Marchand balun has a pair of quarter-wavelength coupled lines of quarter wavelength for dividing and combining signals with the same amplitude and opposite phase.

Creation

Syntax

```
balun = balunMarchand
balun = balunMarchand(Name=Value)
```

Description

`balun = balunMarchand` creates a Marchand balun in the microstrip form with default properties for a resonant frequency of 2.2 GHz.

`balun = balunMarchand(Name=Value)` sets "Properties" on page 1-42 using one or more name-value arguments. For example, `balunMarchand(OutputLineLength=0.016)` creates a Marchand balun with an output line length of 0.016 meters. Properties not specified retain their default values.

Properties

CoupledLineLength — Length of coupled line

0.0178 (default) | positive scalar

Length of the coupled line in meters, specified as a positive scalar.

Example: `balun = balunMarchand(CoupledLineLength=0.0254)`

Data Types: double

CoupledLineWidth — Width of coupled line

0.0003 (default) | positive scalar

Width of the coupled line in meters, specified as a positive scalar.

Example: `balun = balunMarchand(CoupledLineWidth=0.0005)`

Data Types: double

CoupledLineSpacing — Spacing between coupled lines

1.5e-04 (default) | positive scalar

Spacing between the coupled lines in meters, specified as a positive scalar.

Example: `balun = balunMarchand(CoupledLineSpacing=0.00014)`

Data Types: double

OutputLineLength — Length of output line

0.016 (default) | positive scalar

Length of the output line in meters, specified as a positive scalar.

Example: `balun = balunMarchand(OutputLineLength=0.0224)`

Data Types: double

OutputLineWidth — Width of output line

2.9e-04 (default) | positive scalar

Width of the output line in meters, specified as a positive scalar.

Example: `balun = balunMarchand(OutputLineWidth=0.000253)`

Data Types: double

Height — Height of Marchand balun from ground plane

0.0016 (default) | positive scalar

Height of the Marchand balun from the ground plane in meters, specified as a positive scalar.

In the case of a multilayer substrate, you can use the `Height` property to create a Marchand balun where the two dielectrics interface.

Example: `balun = balunMarchand(Height=0.022)`

Data Types: double

Substrate — Type of dielectric material

dielectric object

Type of dielectric material used as a substrate, specified as a `dielectric` object. The default height of the substrate is 0.0013 meters. The dielectric material in a `balunMarchand` object with default properties is FR4.

Example: `d = dielectric("RTDuriod"); balun = balunMarchand(Substrate=d)`

Conductor — Type of metal used in conducting layers

`metal` object

Type of metal used in the conducting layers, specified as a `metal` object. The type of metal in a `balunMarchand` object with default properties is PEC.

Example: `m = metal("Copper"); balun = balunMarchand(Conductor=m)`

Data Types: `string` | `char`

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>designCoupledLine</code>	Calculate dimensions of coupled-line section for specified frequency
<code>designUncoupledLine</code>	Calculate dimensions of uncoupled-line section for specified frequency
<code>designOutputLine</code>	Calculate dimensions of output line section for specified frequency
<code>feedCurrent</code>	Calculate current at feed port
<code>layout</code>	Plot all metal layers and board shape
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>shapes</code>	Extract all metal layer shapes of PCB component
<code>show</code>	Display PCB component structure or PCB shape
<code>sparameters</code>	Calculate S-parameters for RF PCB objects

Examples

Default Marchand Balun

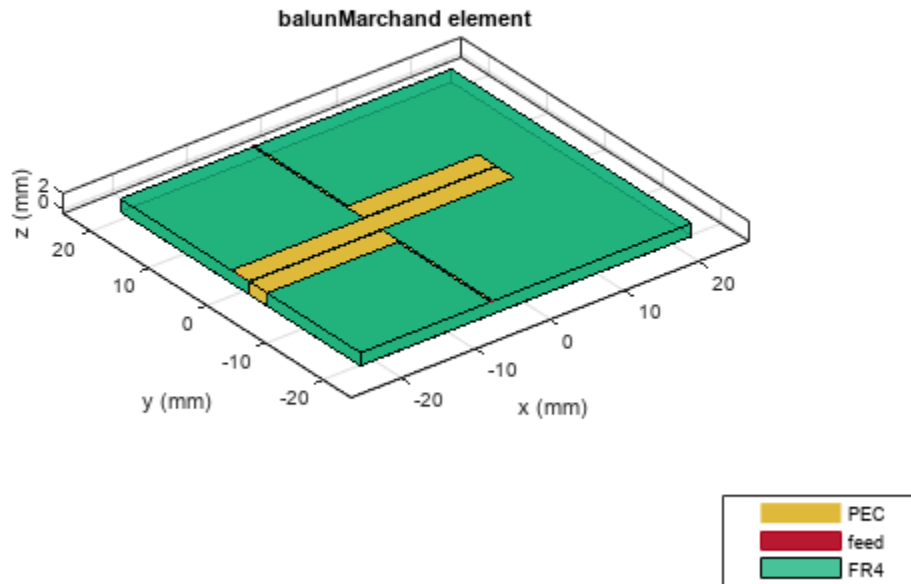
Create a default Marchand balun.

```
balun = balunMarchand
```

```
balun =
  balunMarchand with properties:
    CoupledLineLength: 0.0178
    CoupledLineWidth: 0.0030
    CoupledLineSpacing: 1.5000e-04
    OutputLineLength: 0.0160
    OutputLineWidth: 2.9000e-04
    Height: 0.0016
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

View the balun.

```
show(balun)
```



Multilayer Dielectric Marchand Balun

Create a Marchand balun.

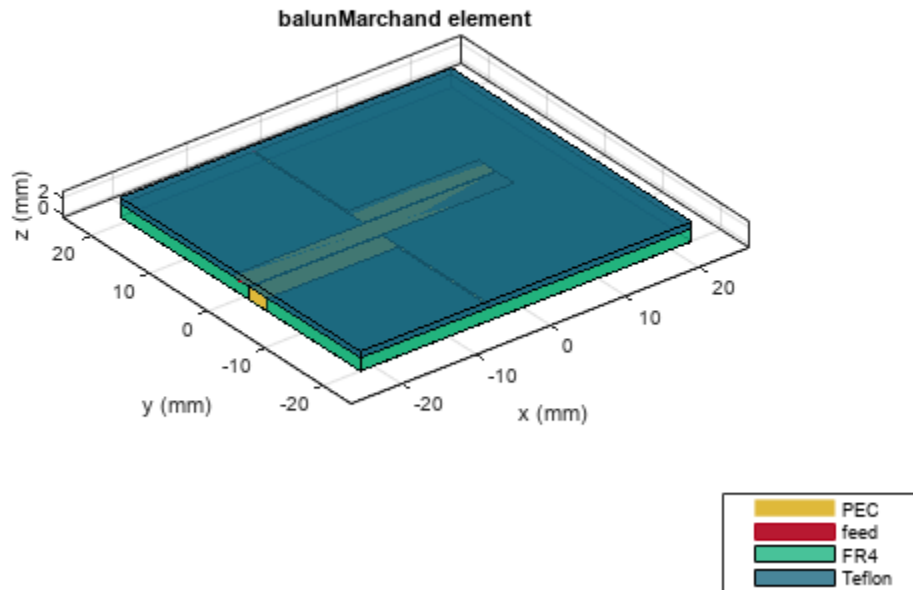
```
balun = balunMarchand;  
balun.Height = 0.0016;
```

Create a multilayer dielectric of FR4 and Teflon.

```
sub = dielectric('FR4','Teflon');  
sub.Thickness = [0.0016 0.0008];  
balun.Substrate = sub;  
figure;
```

View the balun.

```
show(balun);
```

Version History

Introduced in R2022b

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

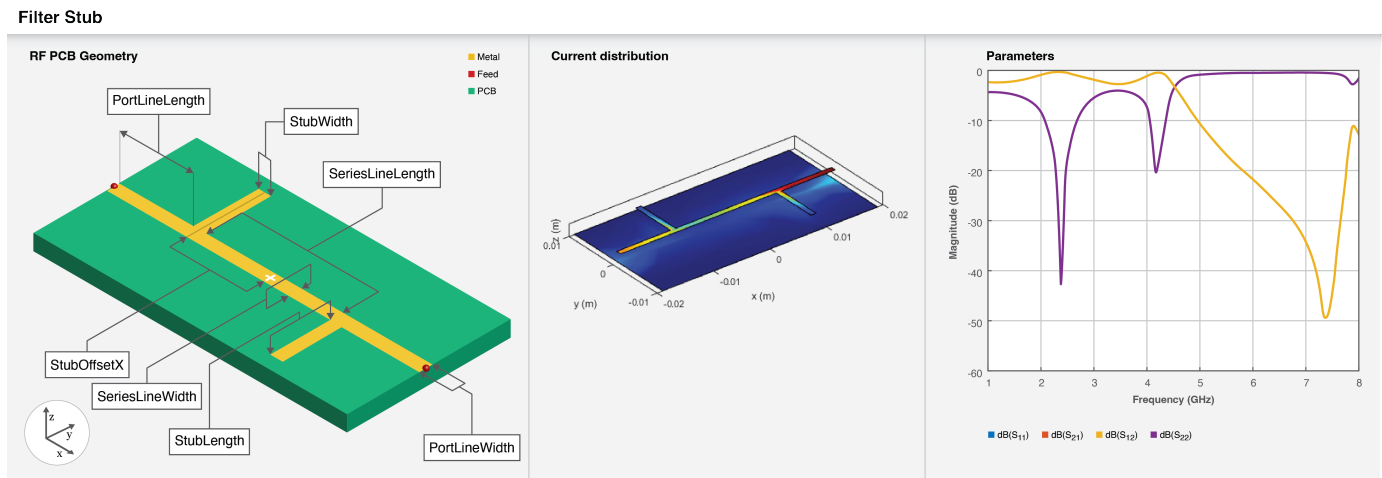
balunCoupledLine

filterStub

Create stub filter in microstrip form

Description

Use the `filterStub` object to create a stub filter in the microstrip form.



Creation

Syntax

```
filter = filterStub
filter = filterStub(Name=Value)
```

Description

`filter = filterStub` creates a default stub using a Teflon substrate. The default passband of the filter is centered around 2 GHz.

`filter = filterStub(Name=Value)` sets “Properties” on page 1-286 using one or more name-value arguments. For example, `filterStub(PortLineLength=9e-4)` creates a stub filter with a port line length of $9e-4$ meters. Properties not specified retain their original values.

Properties

PortLineLength — Length of input and output lines

0.0150 (default) | positive scalar

Length of the input and output lines in meters, specified as a positive scalar.

Example: `filter = filterStub(PortLineLength=0.0014)`

Data Types: double

PortLineWidth — Width of input and output lines

9e-4 (default) | positive scalar

Width of the input and output lines in meters, specified as a positive scalar.

Example: `filter = filterStub(PortLineWidth=8e-4)`

Data Types: double

SeriesLineLength — Length of series line

0.0080 (default) | positive scalar | two-element vector

Length of the series line in meters, specified as a positive scalar or a two-element vector.

Example: `filter = filterStub(SeriesLineLength=0.0070)`

Data Types: double

SeriesLineWidth — Width of series line

9e-4 (default) | positive scalar | two-element vector

Width of the series line in meters, specified as a positive scalar or a two-element vector.

Example: `filter = filterStub(SeriesLineWidth=8e-4)`

Data Types: double

StubLength — Length of stubs

[0.0083 0.0083] (default) | positive scalar | two-element vector

Length of the stubs in meters, specified as a positive scalar or a two-element vector.

Example: `filter = filterStub(StubLength=[0.0078 0.0078])`

Data Types: double

StubWidth — Width of stubs

[9e-04 9e-04] (default) | positive scalar | two-element vector

Width of the stubs in meters, specified as a positive scalar or a two-element vector.

Example: `filter = filterStub(StubWidth=[0.0078 0.0078])`

Data Types: double

StubFeedOffsetX — X-offset from origin

[-0.0090 0.0090] (default) | positive scalar | two-element vector

X-offset from the origin in meters, specified as a positive scalar or a two-element vector.

Example: `filter = filterStub(StubFeedOffsetX=0.0087)`

Data Types: double

StubDirection — Direction of stubs

[1 0] (default) | 1 | 0

Direction of the stubs, specified as a two-element vector of:

- 1 — Indicates that the stub points to the top.
- 0 — Indicates that the stub points to the bottom.

Example: `filter = filterStub(StubDirection=[0 1])`

Data Types: double

StubShort — Flag to indicate short-circuited stubs

[0 0] (default) | 1 | 0

Flag to indicate short-circuited stubs, specified as a two-element vector of:

- 1 — Indicates stub is short-circuited.
- 0 — Indicates stub is open-circuited.

Example: `filter = filterStub(StubShort=[1 1])`

Data Types: double

Height — Height of stub filter from ground plane

0.0016 (default) | positive scalar

Height of the stub filter from the ground plane in meters, specified as a positive scalar. For multilayer dielectrics, use the `Height` property to create the filter between the two dielectric layers.

Example: `filter = filterStub(Height=0.0028)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0200 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `filter = filterStub(GroundPlaneWidth=0.0048)`

Data Types: double

Substrate — Type of dielectric material

dielectric object

Type of dielectric material used as a substrate, specified as a `dielectric` object. The dielectric material in a `filterStub` object with default properties is Teflon.

Example: `d = dielectric("FR4"); filter = filterStub(Substrate=d)`

Conductor — Type of metal used in conducting layers

metal object

Type of metal used in the conducting layers, specified as a `metal` object. The type of metal in a `filterStub` object with default properties is PEC.

Example: `m = metal("Copper"); filter = filterStub(Conductor=m)`

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution

feedCurrent	Calculate current at feed port
getZ0	Calculate characteristic impedance of transmission line
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples

Short-Circuited Stub Filter

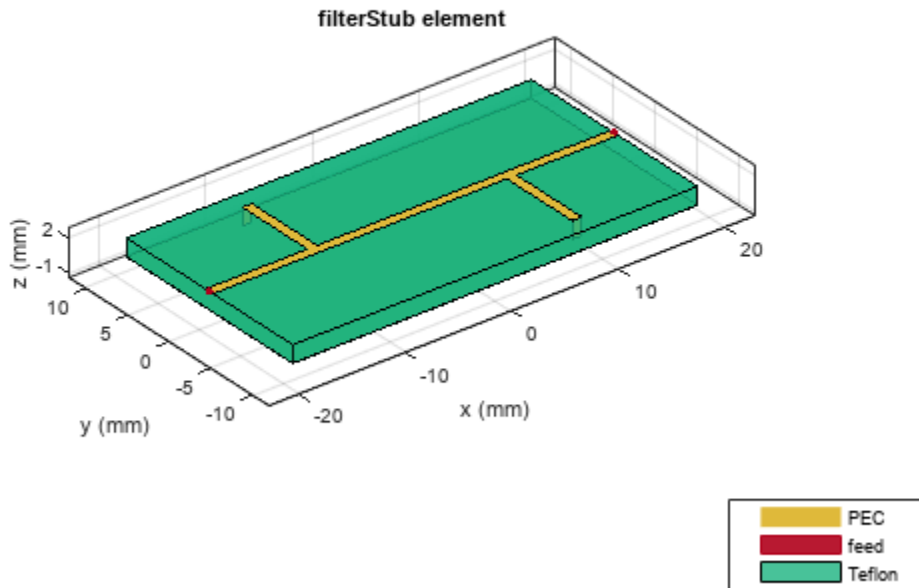
Create a short-circuited stub filter.

```
filter = filterStub(StubShort=[1 1])

filter =
  filterStub with properties:
    PortLineLength: 0.0150
    PortLineWidth: 9.0000e-04
    SeriesLineLength: 0.0080
    SeriesLineWidth: 9.0000e-04
    StubLength: [0.0083 0.0083]
    StubWidth: [9.0000e-04 9.0000e-04]
    StubOffsetX: [-0.0090 0.0090]
    StubDirection: [1 0]
    StubShort: [1 1]
    Height: 0.0016
    GroundPlaneWidth: 0.0200
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

Visualize the filter.

```
show(filter)
```



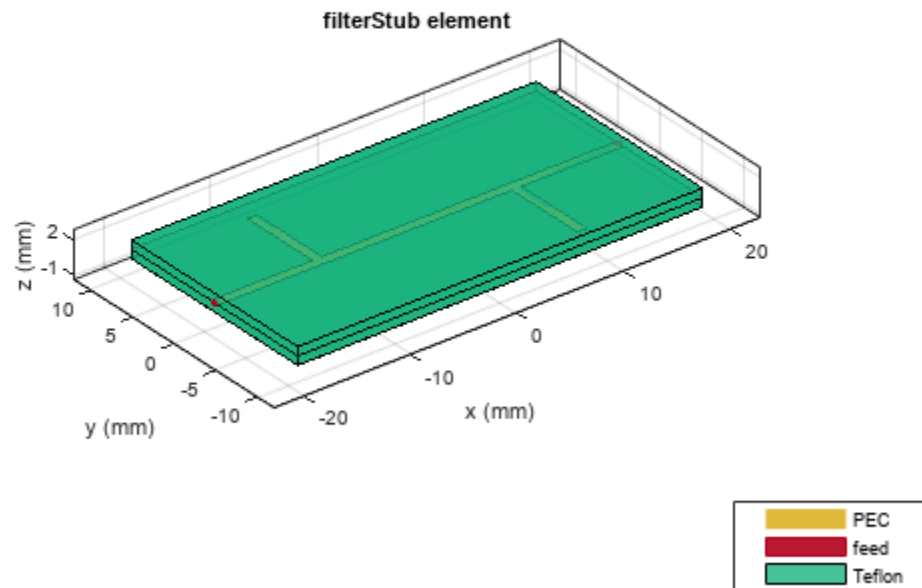
Multilayered Stub Filter

Create a multilayered dielectric stub filter.

```
filter = filterStub;  
filter.Substrate = dielectric(Name={'Teflon', 'Teflon'}, EpsilonR=[2.1 2.1], ...  
    LossTangent=[0 0], Thickness=[0.8e-3 0.8e-3]);  
filter.Height = 0.8e-3;
```

Visualize the filter.

```
show(filter);
```



Version History

Introduced in R2022b

References

- [1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.
- [2] Wen-Hua Tu and Kai Chang. "Compact Microstrip Bandstop Filter Using Open Stub and Spurline." *IEEE Microwave and Wireless Components Letters* 15, no. 4 (April 2005): 268–70. <https://doi.org/10.1109/LMWC.2005.845739>.

See Also

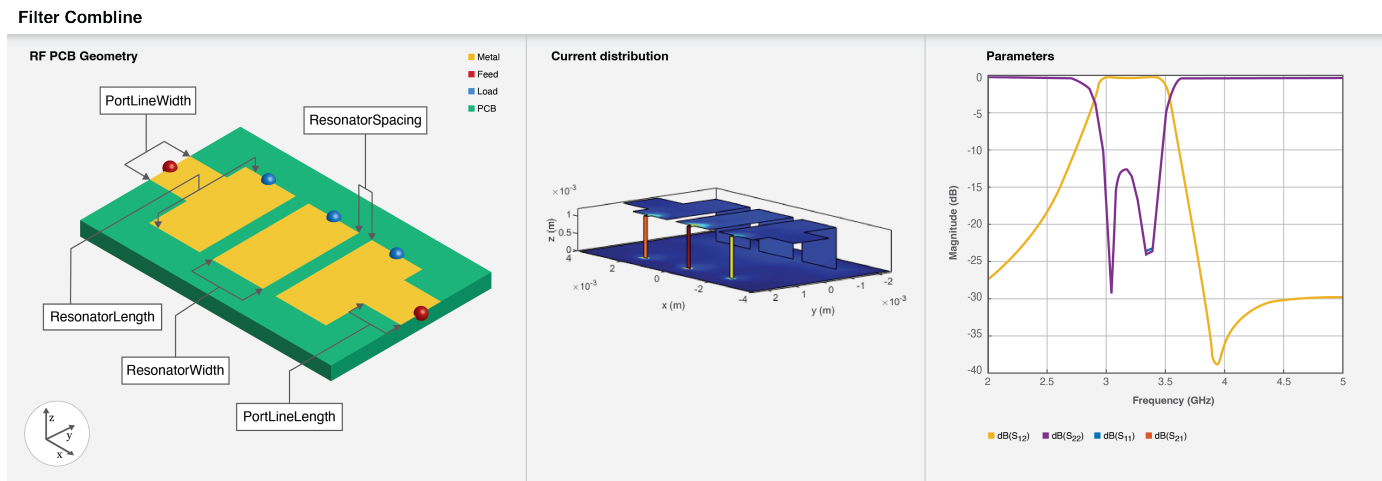
filterHairpin | filterStepImpedanceLowPass

filterComblne

Create combline filter in microstrip form

Description

Use the `filterComblne` object to create a combline filter in the microstrip form.



Creation

Syntax

```
filter = filterComblne
filter = filterComblne(Name=Value)
```

Description

`filter = filterComblne` creates a default combline line filter using an Alumina substrate with the passband of the filter centered around 3.1 GHz.

`filter = filterComblne(Name=Value)` sets “Properties” on page 1-292 using one or more name-value arguments. For example, `filterComblne(ResonatorWidth=0.0016)` creates a combline filter with a resonator width of 0.0016 meters. Properties not specified retain their original values.

Properties

FilterOrder — Order of filter

3 (default) | positive scalar

Order of the filter, specified as a positive scalar.

Example: `filter = filterComblne(FilterOrder=4)`

Data Types: double

ResonatorLength — Length of resonator

0.0030 (default) | positive scalar | vector

Length of the resonator in meters, specified as a positive scalar or a vector of positive elements equal in size to `FilterOrder`.

Example: `filter = filterComblne(ResonatorLength=0.007)`

Data Types: double

ResonatorWidth — Width of resonator

0.0016 (default) | positive scalar

Width of the resonator in meters, specified as a positive scalar.

Example: `filter = filterComblne(ResonatorWidth=0.0017)`

Data Types: double

ResonatorSpacing — Spacing between resonators

4e-04 (default) | positive scalar | vector

Spacing between the resonators in meters, specified as a positive scalar or a vector of positive elements equal in size to `(FilterOrder-1)`.

Example: `filter = filterComblne(ResonatorSpacing=0.0008)`

Data Types: double

ResonatorOffset — Y-offset of each resonator

0 (default) | positive scalar | vector

Y-offset for each resonator in meters, specified as a positive scalar or a vector of positive elements equal in size to `FilterOrder`. If the value is a scalar, all the resonators have the same offset along the Y-axis.

Example: `filter = filterComblne(ResonatorOffset=(0 0.1 0.2))`

Data Types: double

PortLineLength — Length of input and output lines

0.0011 (default) | positive scalar

Length of the input and output lines in meters, specified as a positive scalar.

Example: `filter = filterComblne(PortLineLength=0.0014)`

Data Types: double

PortLineWidth — Width of input and output lines

0.0013 (default) | positive scalar

Width of the input and output lines in meters, specified as a positive scalar.

Example: `filter = filterComblne(PortLineWidth=0.0087)`

Data Types: double

FeedOffset — Y-offset for input and output lines

3e-04 (default) | positive scalar

Y-offset for the input and output lines in meters, specified as a positive scalar.

Example: `filter = filterComblin(FeedOffset=0.0087)`

Data Types: double

Capacitor — Capacitor value

1e-12 (default) | positive scalar

Capacitor value in farad, specified as a positive scalar.

Example: `filter = filterComblin(Capacitor=2e-12)`

Data Types: double

Height — Height of comblin filter from ground plane

0.0012 (default) | positive scalar

Height of the comblin filter from the ground plane in meters, specified as a positive scalar. For multilayer dielectrics, use the `Height` property to create the filter between the two dielectric layers.

Example: `filter = filterComblin(Height=0.0028)`

Data Types: double

GroundPlaneWidth — Width of ground plane

0.0050 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `filter = filterComblin(GroundPlaneWidth=0.0048)`

Data Types: double

Substrate — Type of dielectric material

dielectric object

Type of dielectric material used as a substrate, specified as a `dielectric` object. The dielectric material in a `filterComblin` object with default properties is Alumina.

Example: `d = dielectric("FR4"); filter = filterComblin(Substrate=d)`

Conductor — Type of metal used in conducting layers

metal object

Type of metal used in the conducting layers, specified as a `metal` object. The type of metal in a `filterComblin` object with default properties is PEC.

Example: `m = metal("Copper"); filter = filterComblin(Conductor=m)`

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>current</code>	Calculate and plot current distribution
<code>design</code>	Design coupled line filter around specified frequency
<code>feedCurrent</code>	Calculate current at feed port

getZ0	Calculate characteristic impedance of transmission line
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples

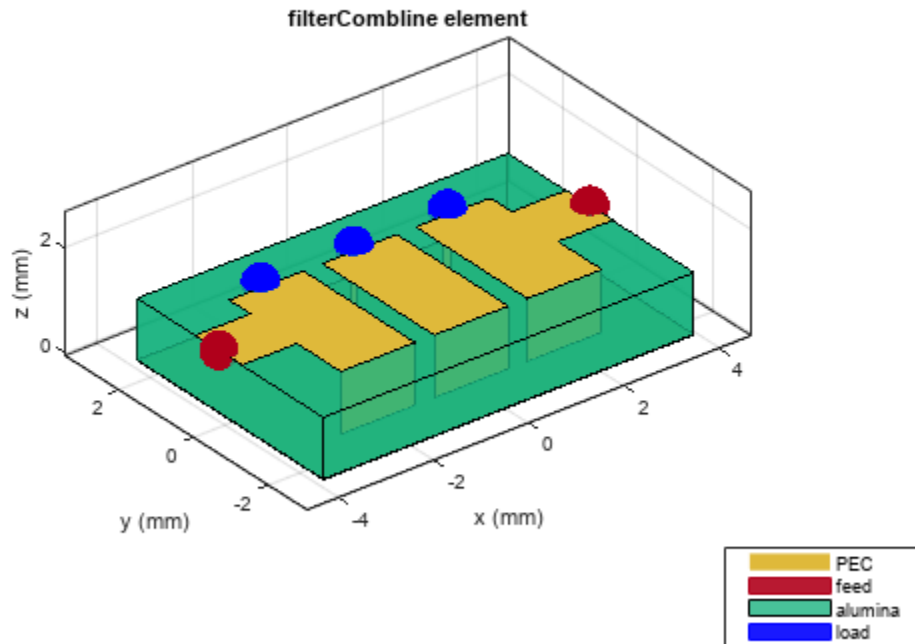
Default Comblines Filter

Create a default comblines filter.

```
filter = filterComblines
filter =
  filterComblines with properties:
    FilterOrder: 3
    ResonatorLength: 0.0030
    ResonatorWidth: 0.0016
    ResonatorSpacing: 4.0000e-04
    ResonatorOffset: 0
    PortLineLength: 0.0011
    PortLineWidth: 0.0013
    FeedOffset: 3.0000e-04
    Capacitor: 1.0000e-12
    Height: 0.0012
    GroundPlaneWidth: 0.0050
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

View the filter.

```
show(filter)
```



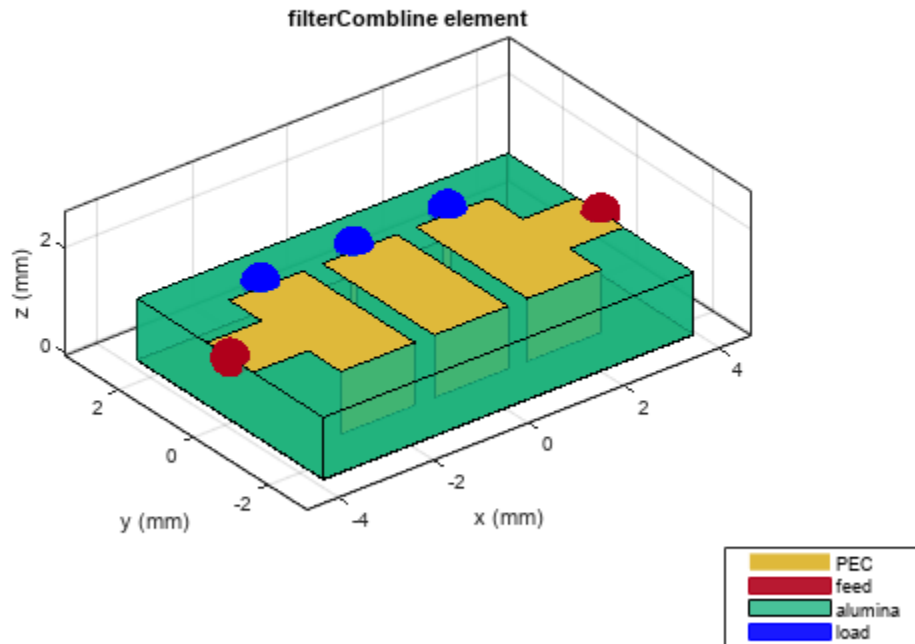
Zero-Offset Feed Comblne Filter

Create a comblne filter with the feed offset equal to zero.

```
filter = filterComblne('FeedOffset',0);
```

Visualize the filter.

```
show(filter);
```



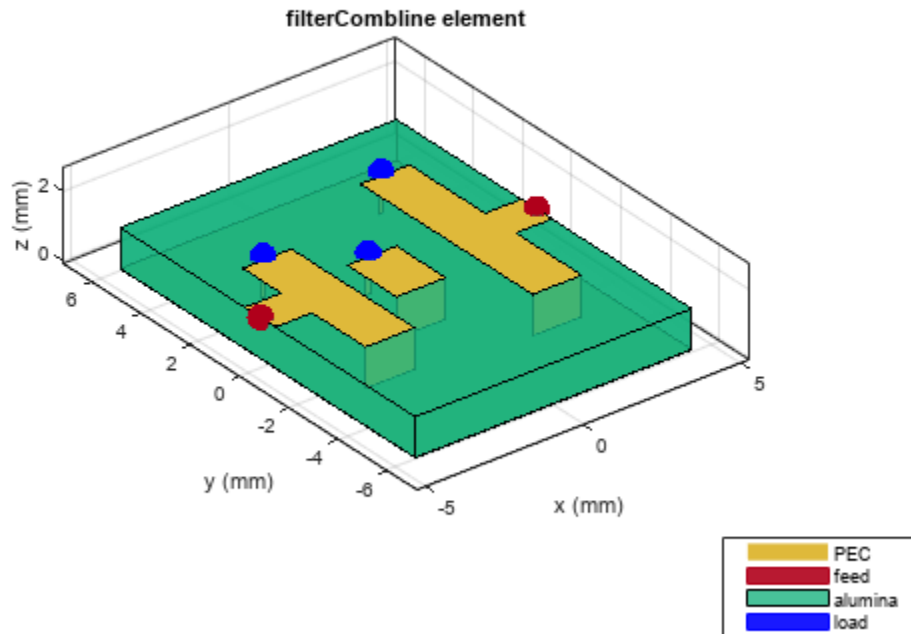
Third-Order Comblne Filter with Resonator Offset

Create a third-order comblne filter with a different resonator offset.

```
filter = filterComblne(FilterOrder=3,ResonatorLength=[0.005,0.002,0.007],...  
    ResonatorSpacing=[0.6e-3,1.2e-3],ResonatorOffset=[0.001e-3,0,0.5e-3], ...  
    GroundPlaneWidth=12e-3);
```

Visualize the filter.

```
show(filter);
```



Version History

Introduced in R2022b

References

- [1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.
- [2] Al-Yasir, Y., R.A. Abd-Alhameed, J.M. Noras, A.M. Abdulkhaleq, and N.O. Parchin. "Design of Very Compact Comblne Band-Pass Filter for 5G Applications." In *Loughborough Antennas & Propagation Conference 2018 (LAPC 2018)*, 61 (4 pp.)-61 (4 pp.). Loughborough, UK: Institution of Engineering and Technology, 2018
- [3] Hong, Jia-Sheng. "Microstrip Filters for RF/Microwave Applications". 2nd ed. Wiley Series in *Microwave and Optical Engineering*. Hoboken, N.J: Wiley, 2011.

See Also

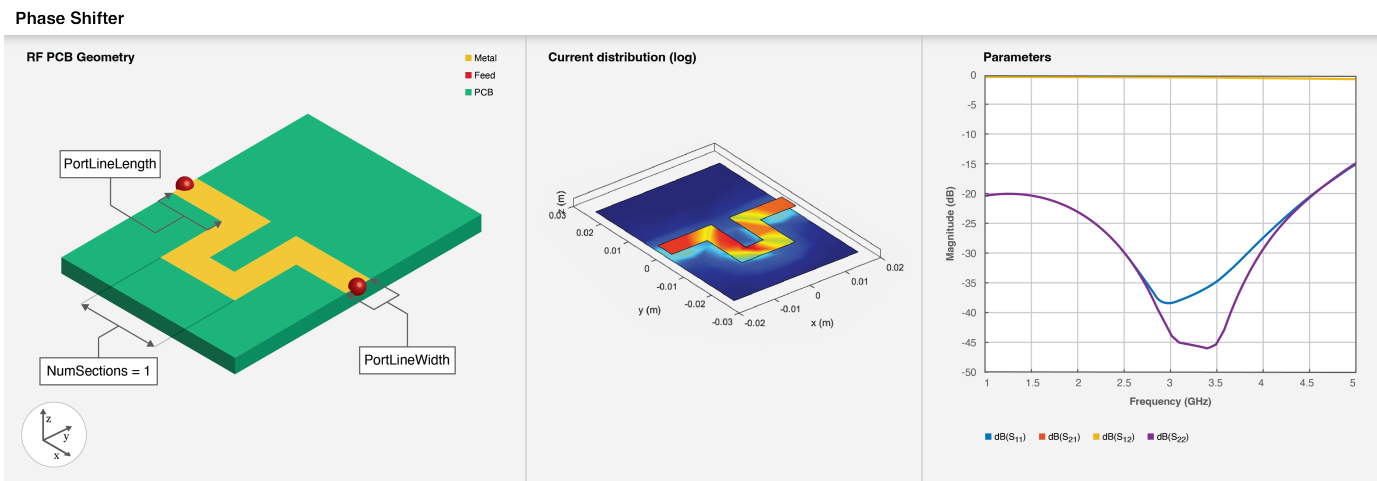
filterHairpin | filterStepImpedanceLowPass

phaseShifter

Create phase shifter in microstrip form

Description

Use the `phaseShifter` object to create a phase shifter in the microstrip form. Phase shifters change the phase of a signal using proper matching at the input and output ports and minimum insertion loss.



Creation

Syntax

```
ps = phaseShifter
ps = phaseShifter(Name=Value)
```

Description

`ps = phaseShifter` creates a phase shifter using Teflon as the default substrate.

`ps = phaseShifter(Name=Value)` sets “Properties” on page 1-299 using one or more name-value arguments. For example, `phaseShifter(NumSections=3)` creates a phase shifter with 3 U-sections. Properties not specified retain their default values.

Properties

NumSections — Number of U-sections

1 (default) | positive scalar

Number of U-sections, specified as a positive scalar.

Example: `ps = phaseShifter(NumSections=3)`

Data Types: double

SectionShape — Shape of U-sections

`ubendRightAngle` (default) | `ubendMitered` | `ubendCurved`

Shape of the U-sections, specified as a `ubendRightAngle`, `ubendMitered`, or `ubendCurved` object.

Example: `shape = ubendMitered; ps = phaseShifter(SectionShape=shape)`

Data Types: double

PortLineLength — Length of port line

`0.0200` (default) | positive scalar

Length of the port line in meters, specified as a positive scalar.

Example: `ps = phaseShifter(PortLineLength=0.0400)`

Data Types: double

PortLineWidth — Width of port line

`0.0050` (default) | positive scalar

Width of the port line in meters, specified as a positive scalar.

Example: `ps = phaseShifter(PortLineWidth=0.0800)`

Data Types: double

Height — Height of phase shifter from ground plane

`0.0016` (default) | positive scalar

Height of the phase shifter from the ground plane in meters, specified as a positive scalar. For multilayer dielectrics, use the `Height` property to create the filter between the two dielectric layers.

Example: `ps = phaseShifter(Height=0.0019)`

Data Types: double

GroundPlaneWidth — Width of ground plane

`0.0300` (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `ps = phaseShifter(GroundPlaneWidth=0.0500)`

Data Types: double

Substrate — Type of dielectric material

dielectric object

Type of dielectric material used as a substrate, specified as a `dielectric` object. The dielectric material in a `phaseShifter` object with default properties is Teflon.

Example: `d = dielectric("FR4"); ps = phaseShifter(Substrate=d)`

Data Types: string | char

Conductor — Type of metal used in conducting layers

'PEC' (default) | metal object

Type of metal used in the conducting layers, specified as a metal object. The type of metal in a phaseShifter object with default properties is PEC.

Example: `m = metal("Copper"); ps = phaseShifter(Conductor=m)`

Data Types: string | char

Object Functions

current	Calculate and plot current distribution
charge	Calculate and plot charge distribution
design	Design phase shifter around specified frequency
feedCurrent	Calculate current at feed port
layout	Plot all metal layers and board shape
mesh	Change and view mesh properties of metal or dielectric in PCB component
shapes	Extract all metal layer shapes of PCB component
show	Display PCB component structure or PCB shape
sparameters	Calculate S-parameters for RF PCB objects

Examples**Default Phase Shifter**

Create a phase shifter with default values.

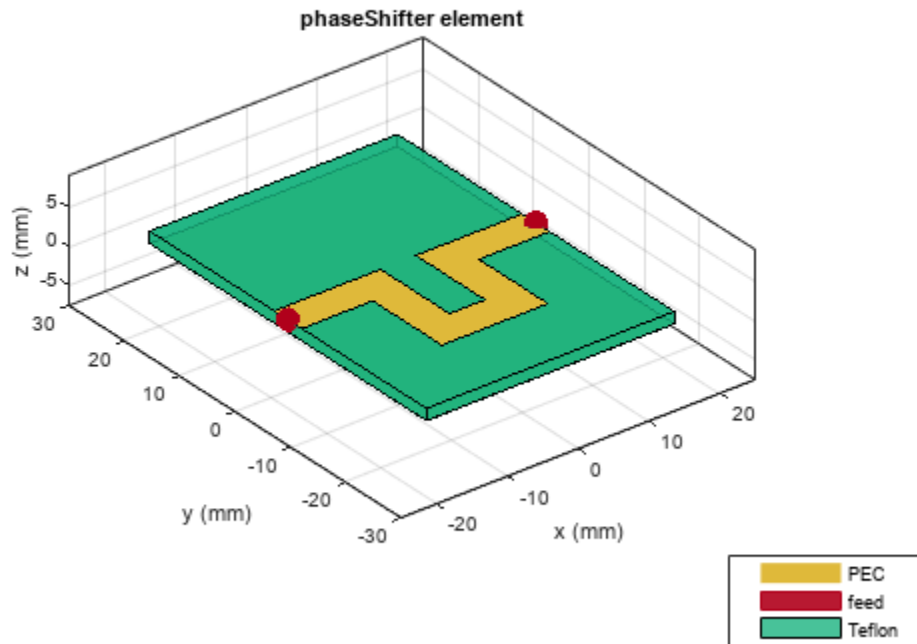
```
ps = phaseShifter
```

```
ps =  
    phaseShifter with properties:
```

```
        NumSections: 1  
        SectionShape: [1x1 ubendRightAngle]  
        PortLineLength: 0.0100  
        PortLineWidth: 0.0050  
            Height: 0.0016  
        GroundPlaneWidth: 0.0500  
            Substrate: [1x1 dielectric]  
            Conductor: [1x1 metal]
```

Visualize the phase shifter.

```
show(ps)
```



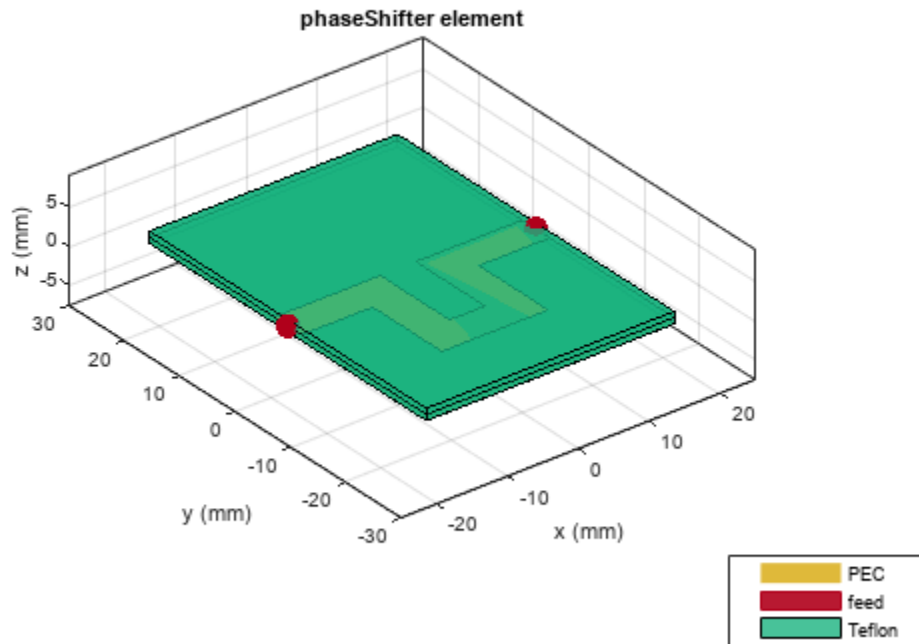
Multilayer Dielectric Phase Shifter

Create a phase shifter at the interface of a multilayered dielectric.

```
ps = phaseShifter;  
ps.Substrate = dielectric(Name={'Teflon', 'Teflon'}, EpsilonR=[2.1 2.1], ...  
    LossTangent=[0 0], Thickness=[0.8e-3 0.8e-3]);  
ps.Height = 0.8e-3;
```

Visualize the phase shifter.

```
show(ps)
```



Version History

Introduced in R2022b

See Also

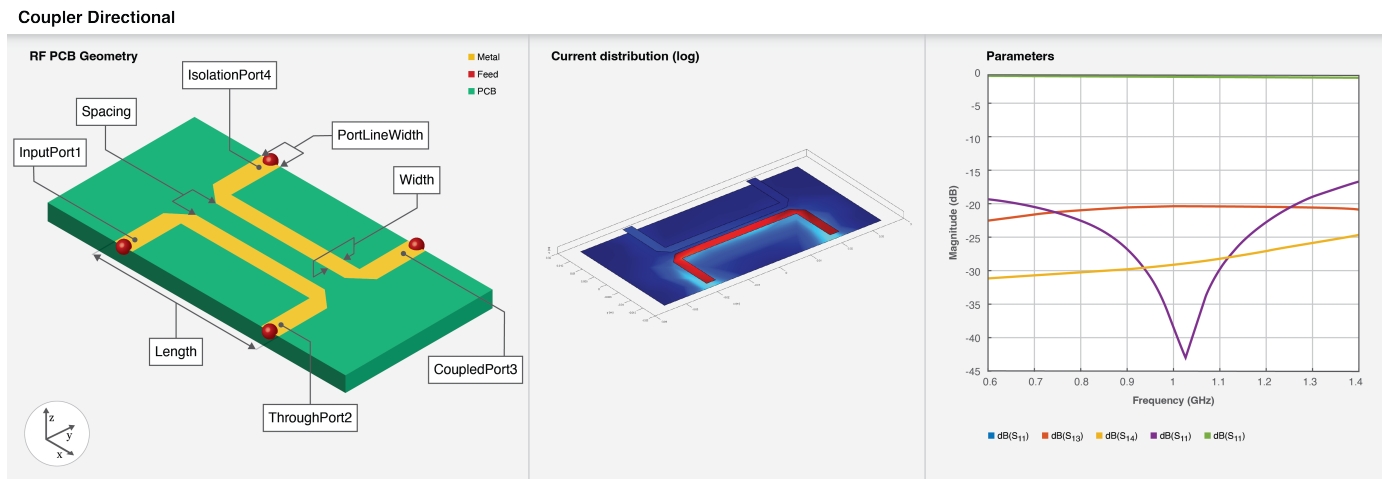
[microstripLine](#) | [wilkinsonSplitter](#)

couplerDirectional

Create single section or multi-section directional coupler in microstrip form

Description

Use the `couplerDirectional` object to create a single section or a multi-section coupler in the microstrip form.



Creation

Syntax

```
coupler = couplerDirectional
coupler = couplerDirectional(Name=Value)
```

Description

`coupler = couplerDirectional` creates a single section coupler in the microstrip form. The default property values are for a resonating frequency of 1 GHz for a 21 dB coupling.

`coupler = couplerDirectional(Name=Value)` sets "Properties" on page 1-304 using one or more name-value arguments. For example, `couplerDirectional(PortLineWidth=0.0286)` creates a directional coupler with a port line width of 0.0286 meters. Properties not specified retain their default values.

Properties

Length — Length of coupled line in each section

0.0397 (default) | positive scalar

Length of the coupled line in each section, specified as a positive scalar.

Example: `coupler = couplerDirectional(Length=0.0096)`

Data Types: double

Width — Width of coupled line in each section

0.0028 (default) | positive scalar | vector

Width of the coupled line in each section, specified as a positive scalar or vector.

Example: `coupler = couplerDirectional(Width=0.0036)`

Data Types: double

Spacing — Spacing between coupled lines in each section

0.0013 (default) | positive scalar | vector

Spacing between the coupled lines in each section, specified as a positive scalar or vector.

Example: `coupler = couplerDirectional(Spacing=0.0016)`

Data Types: double

NumSections — Number of coupled line sections

1 (default) | positive scalar

Number of coupled line sections, specified as a positive scalar. To create a multi-section directional coupler, the number of sections must be greater than one.

Example: `coupler = couplerDirectional(NumSections=2)`

Data Types: double

Corner — Bend type at corners of coupler

'Mitered' (default) | 'Curved' | 'RightAngle'

Bend type at the corners of the coupler for extending the ports, specified as 'Mitered', 'Curved', or 'RightAngle'.

Example: `coupler = couplerDirectional(Corner='Curved')`

Data Types: char

Height — Height of directional coupler from ground plane

0.0016 (default) | positive scalar

Height of the directional coupler from the ground plane in meters, specified as a positive scalar.

In the case of a multilayer substrate, you can use the `Height` property to create a branch line coupler line where the two dielectrics interface.

Example: `coupler = couplerDirectional(Height=0.0026)`

Data Types: double

PortLineWidth — Width of port line

0.0029 (default) | positive scalar

Width of the port line in meters, specified as a positive scalar.

Example: `coupler = couplerDirectional(PortLineWidth=0.0070)`

Data Types: double

GroundPlaneLength — Length of ground plane

0.0697 (default) | positive scalar

Length of the ground plane in meters, specified as a positive scalar.

Example: `coupler = couplerDirectional(GroundPlaneLength=0.046)`

Example: double

GroundPlaneWidth — Width of ground plane

0.0345 (default) | positive scalar

Width of the ground plane in meters, specified as a positive scalar.

Example: `coupler = couplerDirectional(GroundPlaneWidth=0.046)`

Example: double

Substrate — Type of dielectric material

`dielectric(Epsilonr=4.7, LossTangent=0.02, Thickness=0.0016)` (default) | dielectric object

Type of dielectric material used as a substrate, specified as a dielectric object.

Example: `d = dielectric("FR4"); coupler = couplerDirectional(Substrate=d)`

Conductor — Type of metal used in conducting layers

`metal("Copper")` (default) | metal object

Type of metal used in the conducting layers, specified as a metal object.

Example: `m = metal("PEC"); coupler = couplerDirectional(Conductor=m)`

Object Functions

<code>charge</code>	Calculate and plot charge distribution
<code>coupling</code>	Calculate coupling factor of coupler
<code>current</code>	Calculate and plot current distribution
<code>directivity</code>	Calculate directivity of coupler
<code>feedCurrent</code>	Calculate current at feed port
<code>getZ0</code>	Calculate characteristic impedance of transmission line
<code>isolation</code>	Calculate isolation of coupler
<code>layout</code>	Plot all metal layers and board shape
<code>mesh</code>	Change and view mesh properties of metal or dielectric in PCB component
<code>shapes</code>	Extract all metal layer shapes of PCB component
<code>show</code>	Display PCB component structure or PCB shape
<code>sparameters</code>	Calculate S-parameters for RF PCB objects

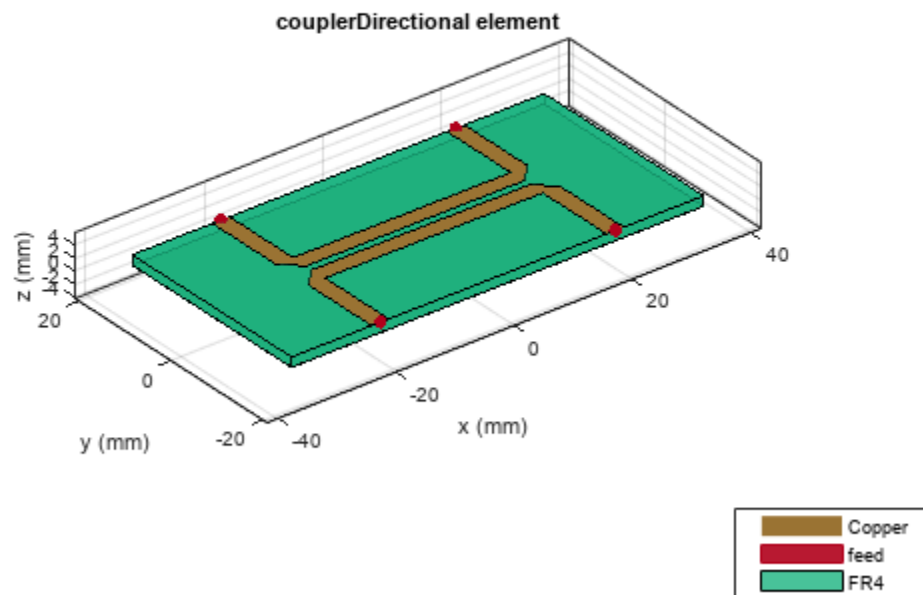
Examples**Default Coupler Directional**

Create a default directional coupler.

```
coupler = couplerDirectional
coupler =
  couplerDirectional with properties:
    Length: 0.0397
    Width: 0.0028
    Spacing: 0.0013
    NumSections: 1
    Corner: 'Mitered'
    Height: 0.0016
    PortLineWidth: 0.0029
    GroundPlaneLength: 0.0697
    GroundPlaneWidth: 0.0345
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

View the coupler.

```
show(coupler)
```



Three-Section Symmetrical Directional Coupler

Create a three-section symmetrical directional coupler on a Teflon substrate.

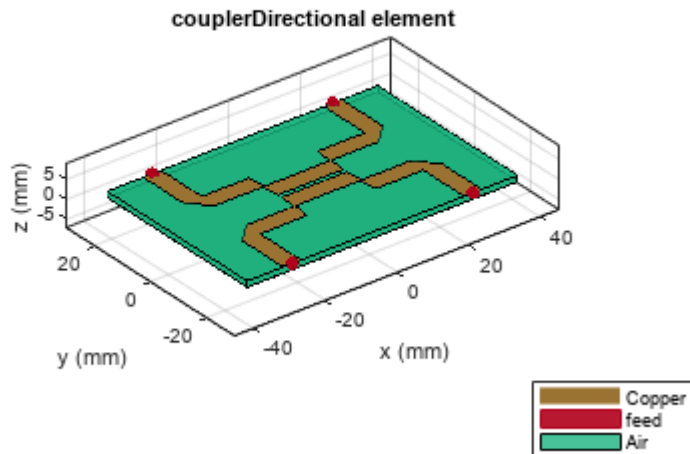
```
coupler = couplerDirectional(Length=18.15e-3, Width=[4.942e-3, ...
4.813e-3, 4.942e-3], Spacing=[10.95e-3, 1.295e-3, 10.95e-3], ...
NumSections=3, Substrate=dielectric(EpsilonR=2.2, LossTangent=0.005, ...
Thickness=0.0016), PortLineWidth=4.942e-3, ...
GroundPlaneLength=75.5e-3, GroundPlaneWidth=50e-3 )
```

```
coupler =
  couplerDirectional with properties:

    Length: 0.0181
    Width: [0.0049 0.0048 0.0049]
    Spacing: [0.0109 0.0013 0.0109]
    NumSections: 3
    Corner: 'Mitered'
    Height: 0.0016
    PortLineWidth: 0.0049
    GroundPlaneLength: 0.0755
    GroundPlaneWidth: 0.0500
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

View the coupler.

```
show(coupler)
```



Version History

Introduced in R2022b

References

[1] Pozar, David M. *Microwave Engineering*. 4th ed. Hoboken, NJ: Wiley, 2012.

See Also

couplerRatrace

Functions

show

Display PCB component structure or PCB shape

Syntax

```
show(pcbcomponent)  
show(shape)
```

Description

`show(pcbcomponent)` displays the PCB component structure in the figure window.

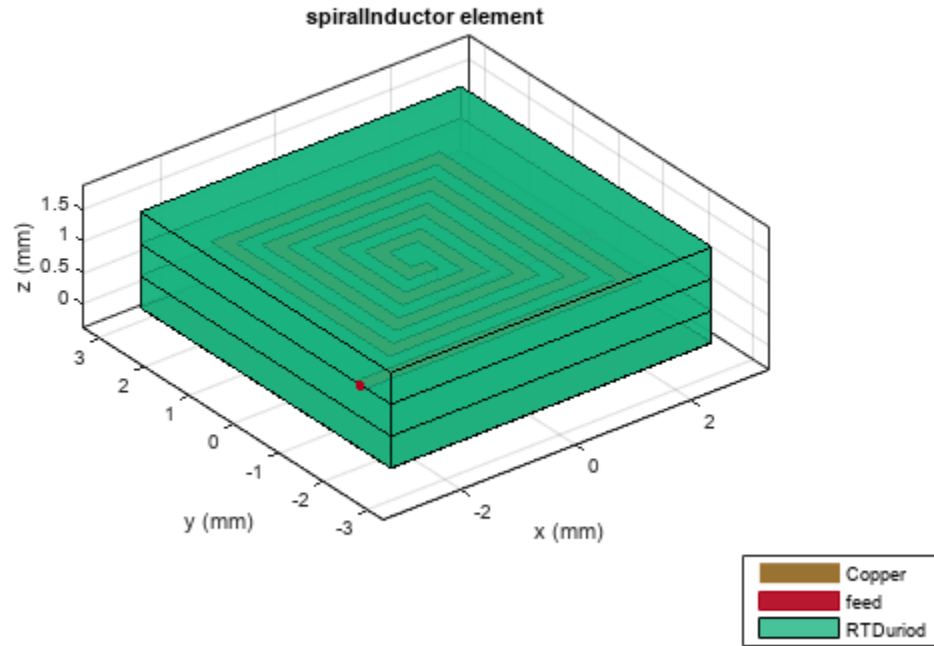
`show(shape)` plots the shape as a filled region using patches.

Examples

Create Default Spiral Inductor

Create and view a default spiral inductor.

```
inductor = spiralInductor  
  
inductor =  
    spiralInductor with properties:  
  
        SpiralShape: 'Square'  
        InnerDiameter: 5.0000e-04  
            Width: 2.5000e-04  
            Spacing: 2.5000e-04  
        NumTurns: 4  
        Height: 0.0010  
        GroundPlaneLength: 0.0056  
        GroundPlaneWidth: 0.0056  
        Substrate: [1x1 dielectric]  
        Conductor: [1x1 metal]  
  
show(inductor)
```



Create Default Curved U-Bend

Create a curved U-bend with default properties.

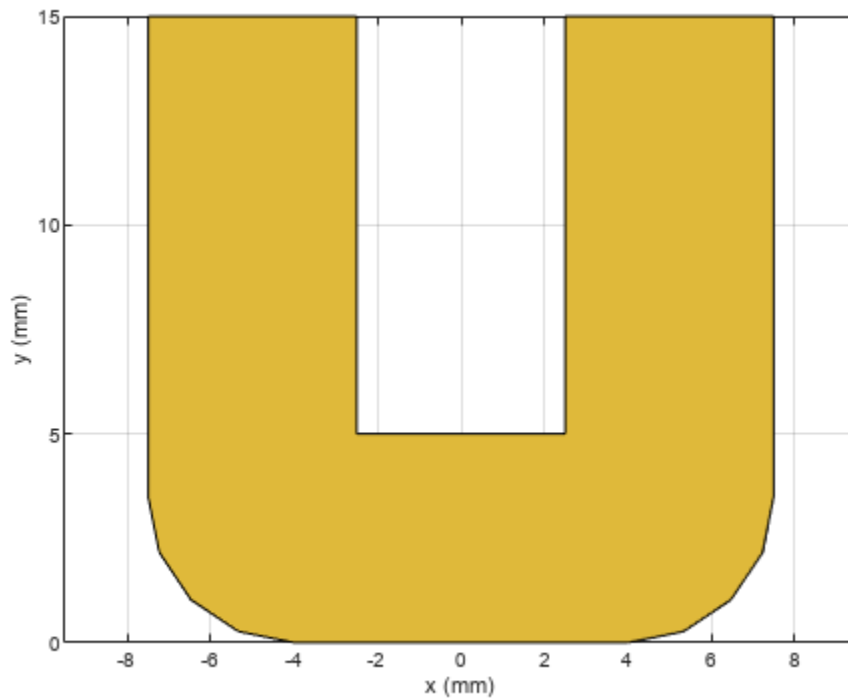
```
curvedubend = ubendCurved
```

```
curvedubend =  
    ubendCurved with properties:
```

```
        Name: 'myCurvedubend'  
    ReferencePoint: [0 0]  
        Length: [0.0150 0.0050 0.0150]  
        Width: [0.0050 0.0050 0.0050]  
    CurveRadius: 0.0035
```

View the shape.

```
show(curvedubend)
```



Input Arguments

pcbcomponent — PCB component object

object handle

PCB component object, specified as a object handle.

Example: `microstrip = microstripLine; show(microstrip)`

shape — Shape created using custom elements and shape objects

object handle

Shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object handle.

Example: `shape = bendCurved; show(shape)`

Version History

Introduced in R2021b

See Also

current

Calculate and plot current distribution

Syntax

```
current(rfpcbobject, frequency)
```

```
i = current(rfpcbobject, frequency)
[i,p] = current(rfpcbobject, frequency)
```

```
current(rfpcbobject, frequency, 'dielectric')
i = current(rfpcbobject, frequency, 'dielectric')
i = current( ____, Name=Value)
```

Description

`current(rfpcbobject, frequency)` calculates and plots the absolute value of the current on the metal surface of a PCB component at the specified frequency.

`i = current(rfpcbobject, frequency)` calculates the x , y , z components of the current on the surface of a PCB component at a specified frequencies.

`[i,p] = current(rfpcbobject, frequency)` returns the current distribution and the points at which the current calculation was performed.

`current(rfpcbobject, frequency, 'dielectric')` calculates and plots the absolute value of the current at the specified frequency on the dielectric surface of the PCB component.

`i = current(rfpcbobject, frequency, 'dielectric')` calculates the x , y , z components of the current on the dielectric surface of a PCB component at the specified frequency.

`i = current(____, Name=Value)` calculates the current on the surface of a PCB component using additional name-value arguments.

Examples

Calculate Current Distribution on Rat-Race Coupler

Create a rat-race coupler with default properties.

```
coupler = couplerRatrace;
```

Set the excitation voltage and the phase angle at the ports of the coupler.

```
v = voltagePort(4)
```

```
v =
    voltagePort with properties:
```

```
    NumPorts: 4
```

```
    FeedVoltage: [1 0 0 0]  
    FeedPhase: [0 0 0 0]  
    PortImpedance: 50
```

```
v.FeedVoltage = [1 0 1 0]
```

```
v =  
    voltagePort with properties:
```

```
    NumPorts: 4  
    FeedVoltage: [1 0 1 0]  
    FeedPhase: [0 0 0 0]  
    PortImpedance: 50
```

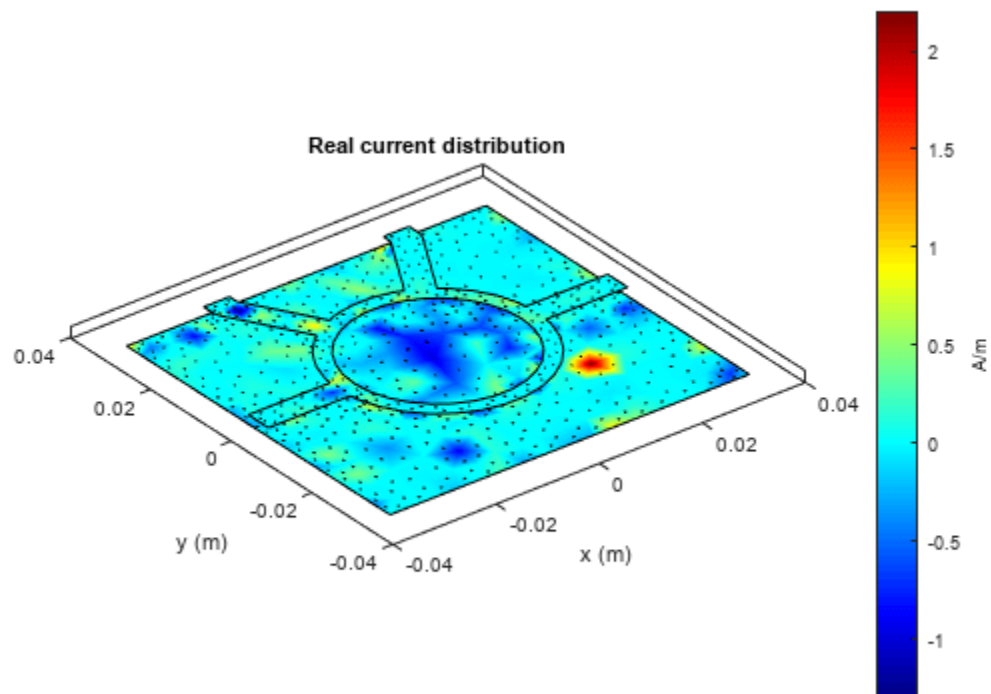
```
v.FeedPhase = [90 0 270 0]
```

```
v =  
    voltagePort with properties:
```

```
    NumPorts: 4  
    FeedVoltage: [1 0 1 0]  
    FeedPhase: [90 0 270 0]  
    PortImpedance: 50
```

Calculate and plot the current on the coupler at 3 GHz.

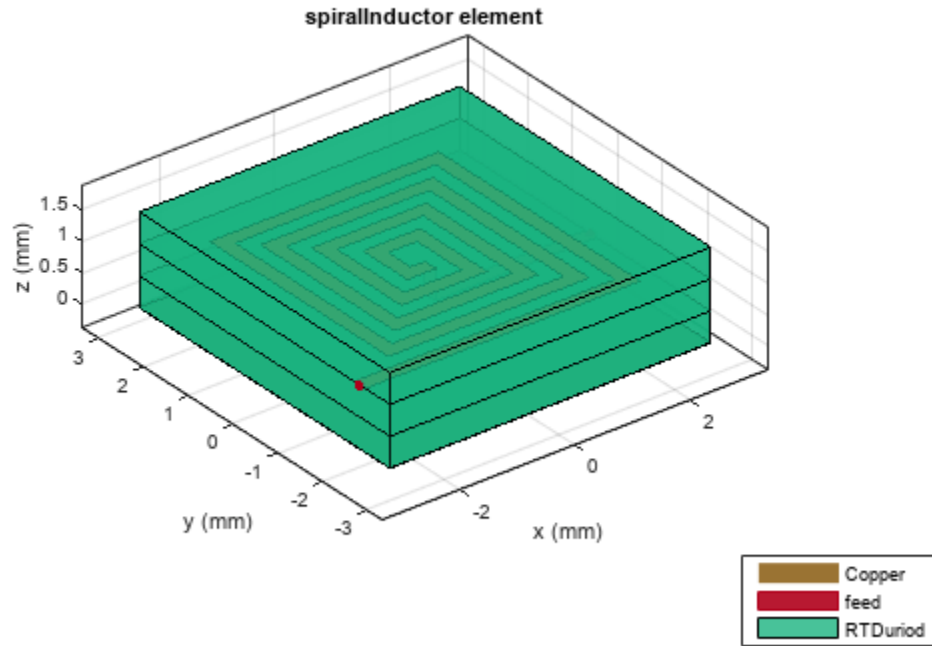
```
figure  
current(coupler,3e9,Excitation=v,Type='real',Direction='on')
```

Calculate Current Distribution on Spiral Inductor

Create a default spiral inductor.

```
inductor = spiralInductor;  
show(inductor)
```



Calculate the current distribution on the inductor at 600 MHz.

```
[i,p] = current(inductor,600e6)
```

i = 3×258 complex

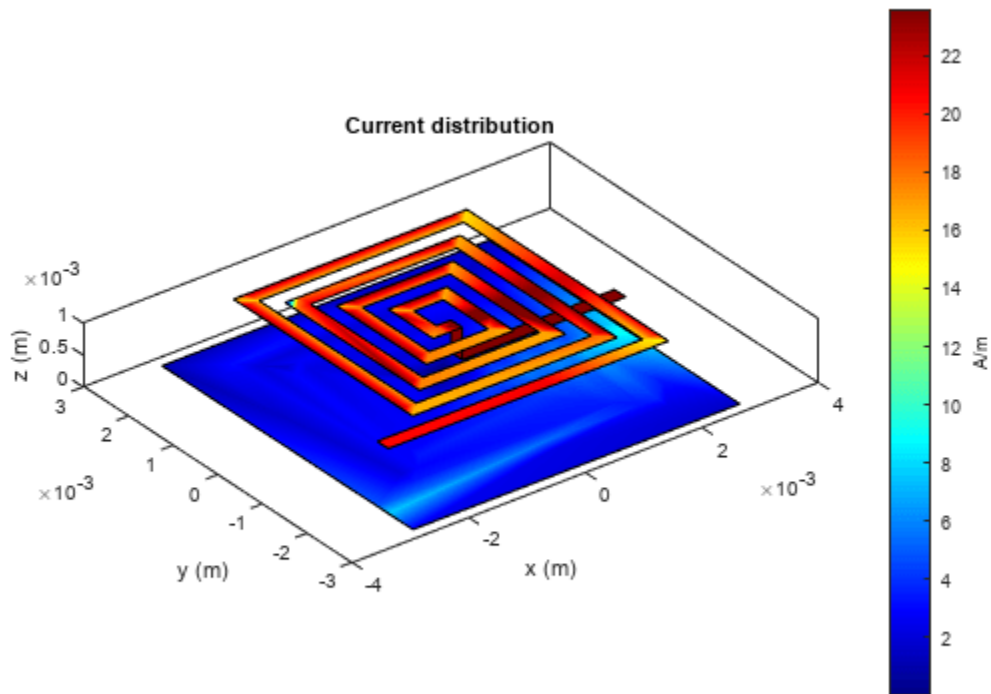
```
-0.0728 + 0.1411i  11.9623 -20.0648i  -0.0003 + 0.0033i  0.1377 - 0.2742i  -0.8881 + 1.3157i
 0.7718 - 1.2051i  3.0732 - 4.7207i  0.8737 - 1.2552i  0.4853 - 0.4545i  -0.0000 + 0.0016i
 0.0000 + 0.0000i  0.0000 + 0.0000i  0.0000 + 0.0000i  0.0000 + 0.0000i  0.0000 + 0.0000i
```

p = 3×258

```
 0.0024  0.0025  0.0025  0.0023  -0.0001  -0.0022  -0.0009  -0.0019  0.0021
-0.0022  -0.0002  0.0017  0.0007  0.0027  0.0025  -0.0022  -0.0021  -0.0020  -0.0000
 0 0 0 0 0 0 0 0 0 0
```

Plot the current distribution.

```
current(inductor,600e6)
```



Input Arguments

rfpcbobject – PCB component object

RF PCB object

PCB component object, specified as an RF PCB object. For a complete list of the PCB components, microstrip bends, and traces, see “PCB Components Catalog” and “Custom Geometry and PCB Fabrication”.

frequency – Frequency used to calculate current distribution

scalar

Frequency to calculate the current distribution, specified as a scalar in Hz.

Example: 70e6

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `scale=log10`

Scale — Scale to visualize current distribution

linear (default) | string scalar | character vector

Scale to visualize the current distribution on the surface of the PCB component, specified as a string or a character vector. The string values are either 'linear', 'log', or 'log10' or as a function. You can specify any mathematical function such as log, log10, cos, or sin.

Data Types: char | function_handle

Slicer — Slicer to focus on desired portion of current plot

'on' | 'off'

Slicer to slice the current plot and focus on the desired portion, specified as 'on' or 'off'. Choosing 'on' adds the slicer panel to the current plot and choosing 'off' does not change the current plot.

Data Types: logical

Excitation — Excitation using voltage source

function handle

Excitation using as voltage source, specified function handle from the voltagePort function.

Data Types: char | function_handle

Type — Choose component of metal current

absolute (default) | real | imaginary

Choose the component to display of the metal current, specified as absolute, real, or imaginary.

Data Types: char | function_handle

Direction — Visualize direction of vector when plotting metal current

off (default) | on

Visualize the direction of the vector when plotting metal current, specified as off or on.

Data Types: logical

Output Arguments**i — x, y, z components of current distribution**

3-by-*n* complex matrix in A/m

x, *y*, *z* components of the current distribution, returned as a 3-by-*n* complex matrix in A/m. The value of the current is calculated on every triangle mesh on the surface of the PCB component.

p — Cartesian coordinates representing center of each triangle in mesh

3-by-*n* real matrix

Cartesian coordinates representing the center of each triangle in the mesh, returned as a 3-by-*n* real matrix.

Version History

Introduced in R2021b

See Also

voltagePort | charge

charge

Calculate and plot charge distribution

Syntax

```
charge(rfpcbobject, frequency)
```

```
c = charge(rfpcbobject, frequency)
[c,p] = charge(rfpcbobject, frequency)
```

```
charge(rfpcbobject, frequency, 'dielectric')
c = charge(rfpcbobject, frequency, 'dielectric')
c = charge( ____, Name=Value)
```

Description

`charge(rfpcbobject, frequency)` calculates and plots the absolute value of the charge in C/m on the metal surface of a PCB component at the specified frequencies.

`c = charge(rfpcbobject, frequency)` calculates a vector of charges in C/m on the metal surface of a PCB component, at the specified frequencies.

`[c,p] = charge(rfpcbobject, frequency)` returns the point at which the charge calculation was performed.

`charge(rfpcbobject, frequency, 'dielectric')` calculates and plots the absolute value of the charge at the specified frequency on the dielectric surface of the PCB component.

`c = charge(rfpcbobject, frequency, 'dielectric')` calculates the charge on the dielectric surface of a PCB component at the specified frequency.

`c = charge(____, Name=Value)` calculates the charge on the surface of a PCB component using additional name-value arguments.

Examples

Calculate Charge Distribution on Rat-Race Coupler

Create a rat-race coupler with default properties.

```
coupler = couplerRatrace;
```

Set the feed voltage and phase at the coupler ports.

```
v = voltagePort(4)
```

```
v =  
voltagePort with properties:
```

```
NumPorts: 4
```

```
FeedVoltage: [1 0 0 0]
FeedPhase: [0 0 0 0]
PortImpedance: 50
```

```
v.FeedVoltage = [1 0 1 0]
```

```
v =
voltagePort with properties:
```

```
NumPorts: 4
FeedVoltage: [1 0 1 0]
FeedPhase: [0 0 0 0]
PortImpedance: 50
```

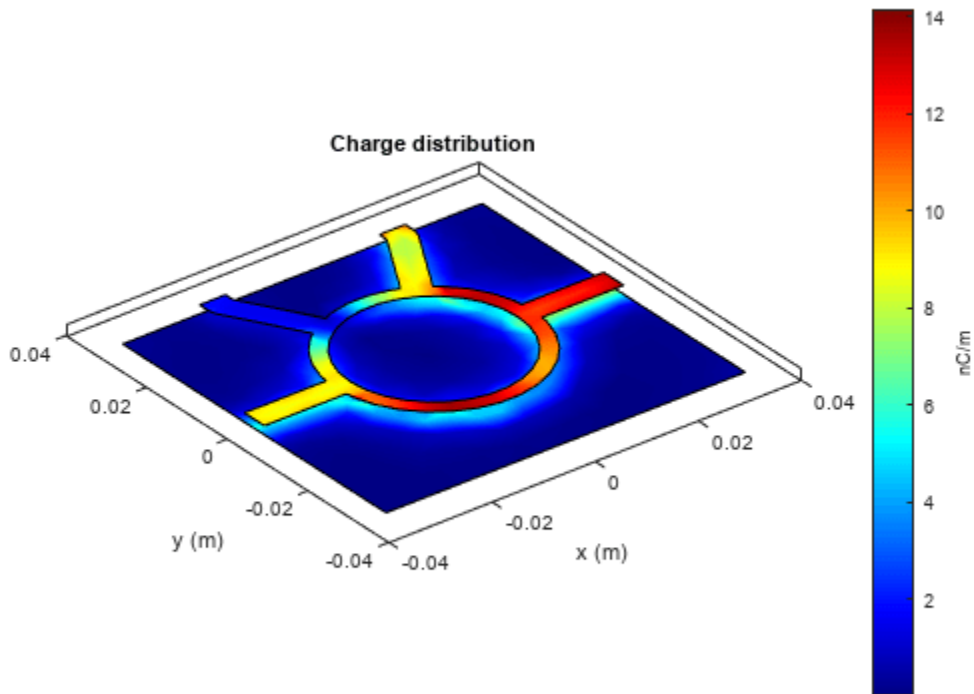
```
v.FeedPhase = [90 0 270 0]
```

```
v =
voltagePort with properties:
```

```
NumPorts: 4
FeedVoltage: [1 0 1 0]
FeedPhase: [90 0 270 0]
PortImpedance: 50
```

Calculate and view the charge distribution of the coupler at 3 GHz.

```
figure
charge(coupler,3e9,Excitation=v)
```



Input Arguments

rffpcbobject — PCB component object

RF PCB object

PCB component object, specified as an RF PCB object. For a complete list of the PCB components and shapes, see “PCB Components Catalog” and “Custom Geometry and PCB Fabrication”.

frequency — Frequency to calculate charge distribution

scalar

Frequency to calculate charge distribution in hertz, specified as a scalar.

Example: `70e6`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `scale='log10'`

Scale — Scale to visualize charge distribution

string (default) | character vector

Scale to visualize the charge distribution on the surface of the PCB component, specified as a string or a character vector. The string values are either 'linear', 'log', or 'log10' or as a function. You can specify any mathematical function such as log, log10, cos, or sin.

Data Types: char | function_handle

Excitation — Excitation using voltage source

function handle

Excitation using as voltage source of N-ports to excite an N-port RF PCB component, specified a string or a function handle.

Data Types: string | function_handle

Output Arguments**c — Complex charges**1-by-*n* vector in C/m

Complex charges, returned as a 1-by-*n* vector in C/m. This value is calculated on every triangle mesh or every dielectric tetrahedron face on the surface of a PCB component.

p — Cartesian coordinates representing center of each triangle in mesh3-by-*n* real matrix

Cartesian coordinates representing the center of each triangle in the mesh, returned as a 3-by-*n* real matrix.

Version History

Introduced in R2021b

See Also

current

feedCurrent

Calculate current at feed port

Syntax

```
feedCurrent(rfpcbobject, frequency)
feedCurrent( ____, Name=Value)
```

Description

`feedCurrent(rfpcbobject, frequency)` calculates the current in A/m at the feed port of a PCB component at the specified frequency. The feed current when multiplied by the PCB impedance gives the voltage across the PCB component.

`feedCurrent(____, Name=Value)` calculates the feed current at the feed port of the PCB component using additional name-value arguments.

Examples

Calculate Feed Current of Rat-Race Coupler

Create a rat-race coupler with default properties.

```
coupler = couplerRatrace;
```

Set the feed voltage and phase at the ports of the rat-race coupler:

```
v = voltagePort(4);
v.FeedVoltage = [1 0 1 0];
v.FeedPhase = [90 0 270,0];
```

Calculate the feed current at 3 GHz.

```
If = feedCurrent(coupler,3e9)
```

```
If = 1x4 complex
```

```
0.0103 + 0.0005i 0.0033 - 0.0061i -0.0001 + 0.0002i -0.0049 + 0.0051i
```

Input Arguments

rfpcbobject — PCB component object

RF PCB object

PCB component object, specified as an RF PCB object. For a complete list of the PCB components and shapes, see “PCB Components Catalog” and “Custom Geometry and PCB Fabrication”.

frequency — Frequency to calculate feed current

scalar | vector

Frequency to calculate the feed current, specified as a scalar integer in Hz or as a vector with each element specified in Hz.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `excitation='2'`

excitation — Excitation using voltage source

string (default) | function handle

Excitation using as voltage source, specified function handle from the `voltagePort` function.

Data Types: `char` | `function_handle`

Version History

Introduced in R2021b

See Also

`current` | `voltagePort`

mesh

Change and view mesh properties of metal or dielectric in PCB component

Syntax

```
mesh(object)
mesh(shape)
mesh( ____,Name,Value)
meshdata = mesh( ____,Name,Value)
```

Description

`mesh(object)` plots the mesh used to analyze a PCB component.

`mesh(shape)` plots the mesh for the shapes.

`mesh(____,Name,Value)` changes and plots the mesh structure of a PCB component, using additional options specified by the name-value pairs. You can also determine the number of unknowns from the number of basis functions in the output.

`meshdata = mesh(____,Name,Value)` returns a mesh structure that specifies the properties used to analyze the PCB component.

Examples

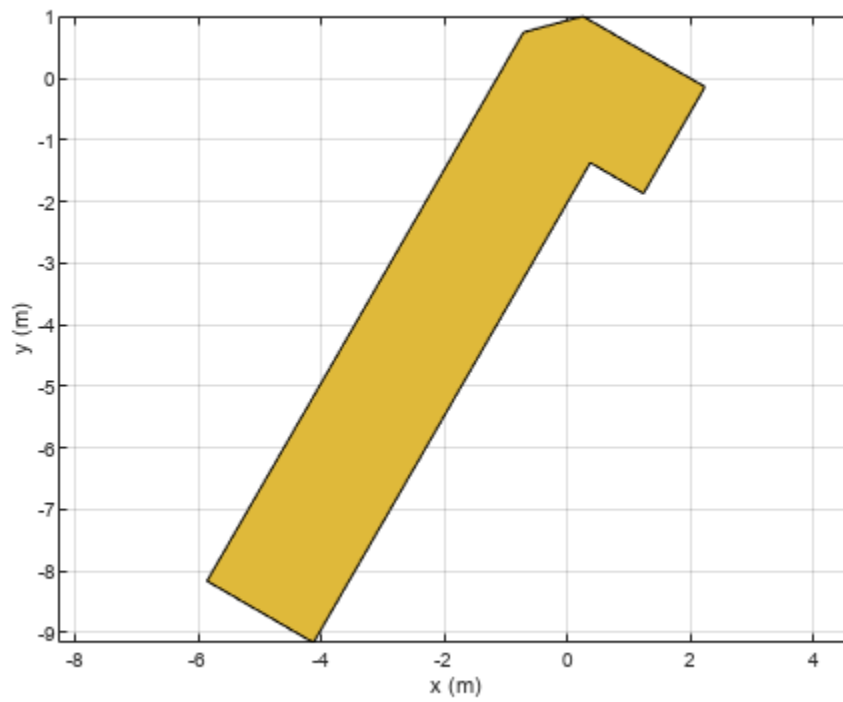
Mesh Rotated Mitered Bend Shape

Create a mitered bend shape of lengths of 10 m and 2 m, width of 2 m, and rotate it about the Z-axis by 60 degrees.

```
bend = bendMitered(Length=[10 2],Width=[2 2],MiterDiagonal=1);
bend = rotateZ(bend,60)
```

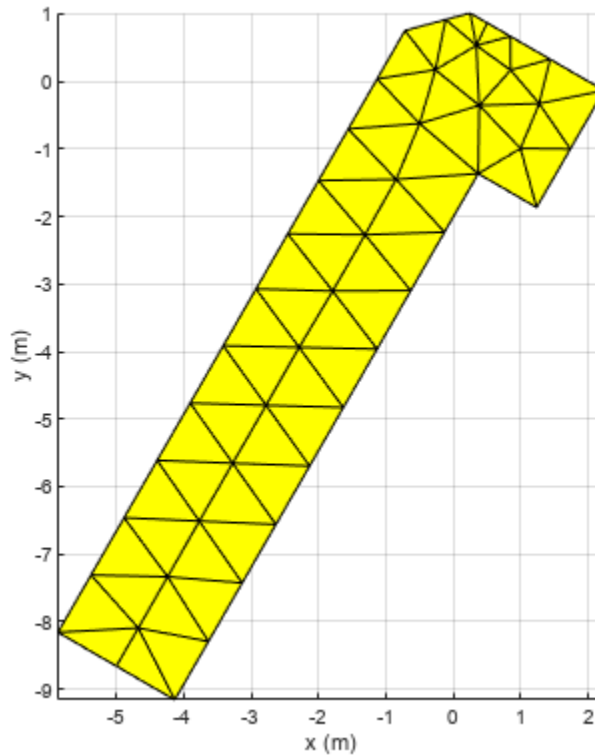
```
bend =
    bendMitered with properties:
        Name: 'myMiteredbend'
    ReferencePoint: [0 0]
        Length: [10 2]
        Width: [2 2]
    MiterDiagonal: 1
```

```
show(bend)
```



Mesh the mitered bend shape at a maximum edge length of 1 m.

```
mesh(bend,MaxEdgeLength=1)
```



Input Arguments

object — PCB component

object handle

PCB component, specified as an object handle. For complete list of PCB components and shapes, see “PCB Components Catalog”

shape — Shape created using custom elements and shape objects

object handle

Shape created using custom elements and shape objects, specified as an object handle. For a complete list of custom shapes, see “Custom Geometry and PCB Fabrication”.

Example: `c = bendCurved; mesh(c)`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` pair arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1`, `Value1`, ..., `NameN`, `ValueN`.

Example: `'MaxEdgeLength', 0.1`

MaxEdgeLength — Maximum edge length of triangles in mesh

scalar

Maximum edge length of triangles in mesh, specified as a positive scalar in meters. All triangles in the mesh have sides less than or equal to the 'MaxEdgeLength' value.

Data Types: double

MinEdgeLength — Smallest edge length of triangles in mesh

scalar

Smallest edge length of triangles in mesh, specified a positive scalar in meters. All triangles in the mesh have sides greater than or equal to the 'MinEdgeLength'.

Data Types: double

GrowthRate — Mesh growth rate

0.7 (default) | scalar

Gradation in the triangle size of the mesh, specified a scalar. The default value of 0.7 states that the growth rate of the mesh is 70 percent. Growth rate values lie between 0 and 1.

Data Types: double

View — Customize mesh view of a PCB component

'all' (default) | 'metal' | 'dielectric surface' | 'dielectric volume'

Customize mesh view of a PCB component, specified as a comma-separated pair consisting of 'View' and 'all', 'metal', 'dielectric surface', or 'dielectric volume'.

You choose 'dielectric surface' to view the boundary triangle mesh of the dielectric. You choose 'dielectric volume' to view the tetrahedral volume mesh of the dielectric.

Data Types: char

Slicer — Option to enable or disable plot interactivity

0 (default) | 1 | 'off' | 'on'

Option to enable or disable plot interactivity, specified as 'on' or 'off', or as numeric or logical 1(true) or 0(false). Set this argument to 1 or 'on' to open the plot with the slicer panel, and to slice and view the desired cross section of the plot along the xy-, yz-, and xz- planes. Set this argument to 0 or 'off' to open the plot without the slicer panel.

Example: Slicer='on'

Data Types: string | logical

Version History

Introduced in R2021b

See Also

meshconfig

getZ0

Calculate characteristic impedance of transmission line

Syntax

```
z0 = getZ0(txline)
z0 = getZ0(txline, freq)
```

Description

`z0 = getZ0(txline)` calculates the characteristic impedance `z0` of a transmission line.

`z0 = getZ0(txline, freq)` calculates the characteristic impedance at the specified frequency.

Examples

Calculate Characteristic Impedance of Microstrip Transmission Line

Create a microstrip transmission line with default properties.

```
mline = microstripLine
mline =
  microstripLine with properties:
        Length: 0.0200
        Width: 0.0050
        Height: 0.0016
  GroundPlaneWidth: 0.0300
        Substrate: [1x1 dielectric]
        Conductor: [1x1 metal]
```

Calculate the characteristic impedance of the line.

```
Z0 = getZ0(mline)
Z0 = 49.6897 - 0.0003i
```

Input Arguments

txline — Transmission line

transmission line object (default)

Transmission line, specified as one of the following: `coplanarWaveguide`, `coupledMicrostripLine`, and `microstripLine`.

Example: `txline = microstripLine; getZ0(txline)`. Calculates the characteristic impedance of the microstrip transmission line object with handle `txline`.

Data Types: char | string

freq – Frequency

positive scalar

Frequency to calculate the characteristic impedance, specified as a positive scalar in hertz.

Output Arguments**z0 – Characteristic impedance of transmission lines**

complex scalar

Characteristic impedance of the transmission line, returned as a complex scalar.

Data Types: double

Version History

Introduced in R2021b

See Also

sparameters

inductance

Calculate inductance

Syntax

```
inductance(object, frequency)  
l = inductance(object, frequency)
```

Description

`inductance(object, frequency)` calculates and plots the inductance of an inductor over the specified frequency.

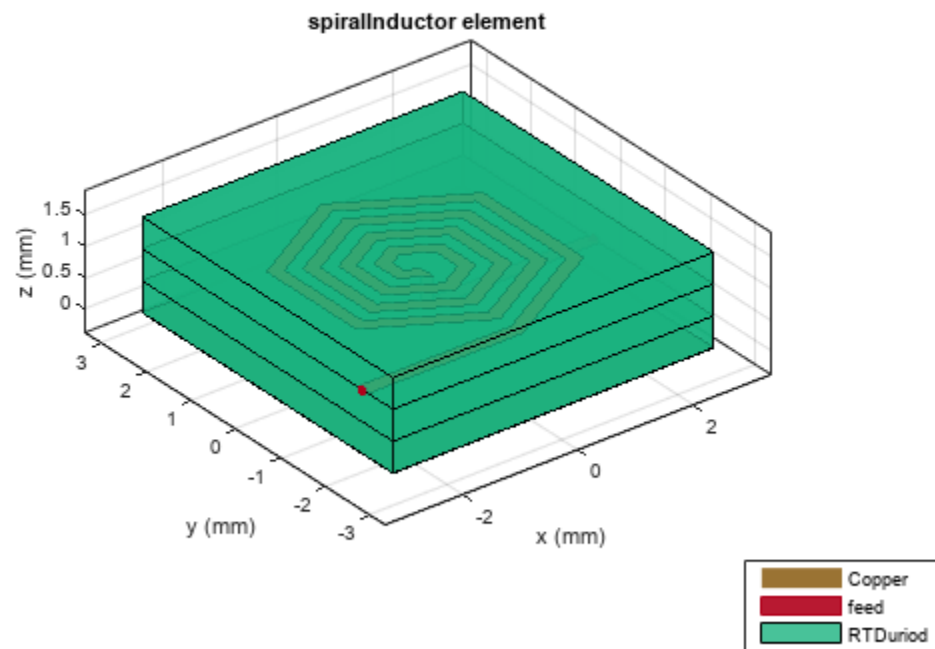
`l = inductance(object, frequency)` calculates the inductance of an inductor over the specified frequency.

Examples

Calculate Inductance of Spiral Inductor

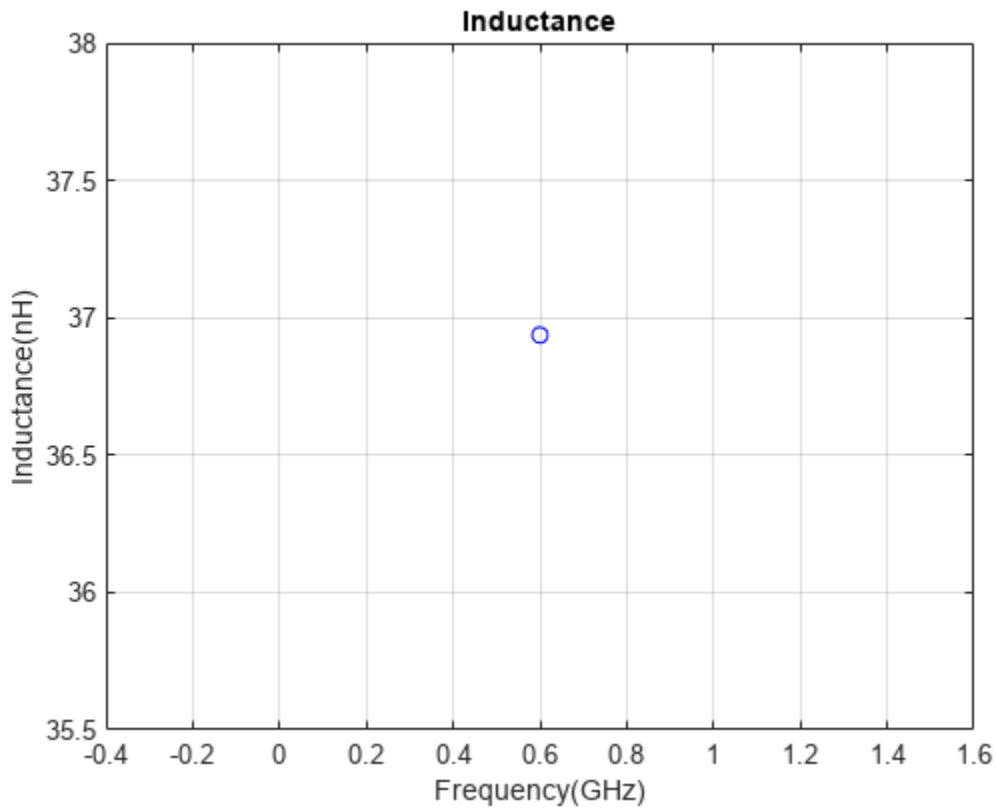
Create a hexagonal spiral inductor with default properties.

```
inductor = spiralInductor(SpiralShape='Hexagon');  
show(inductor)
```



Calculate the inductance of the inductor at 600 MHz.

```
inductance(inductor,600e6)
```



```
l = inductance(inductor,600e6)
```

```
l = 3.6937e-08
```

Input Arguments

object – Spiral inductor

`spiralInductor` object

Spiral inductor, specified as an `spiralInductor` object.

Data Types: `char` | `string`

frequency – Frequency to calculate inductance

`nonnegative scalar` | `nonnegative vector`

Frequency to calculate inductance, specified as a nonnegative scalar or vector in hertz.

Data Types: `double`

Output Arguments

l – Inductance of inductor

`scalar` | `vector`

Inductance of the inductor, returned as a scalar or vector in henries.

Data Types: double

Version History

Introduced in R2021b

See Also

capacitance

capacitance

Calculate capacitance

Syntax

```
capacitance(object, frequency)
c = capacitance(object, frequency)
capacitance( ____, Name=Value)
```

Description

`capacitance(object, frequency)` calculates and plots the capacitance of a capacitor over the specified frequency.

`c = capacitance(object, frequency)` calculates the capacitance of a capacitor over the specified frequency.

`capacitance(____, Name=Value)` calculates the capacitance with additional options specified using name-value arguments.

Examples

Calculate Capacitance of Interdigital Capacitor

Create an interdigital capacitor using default properties.

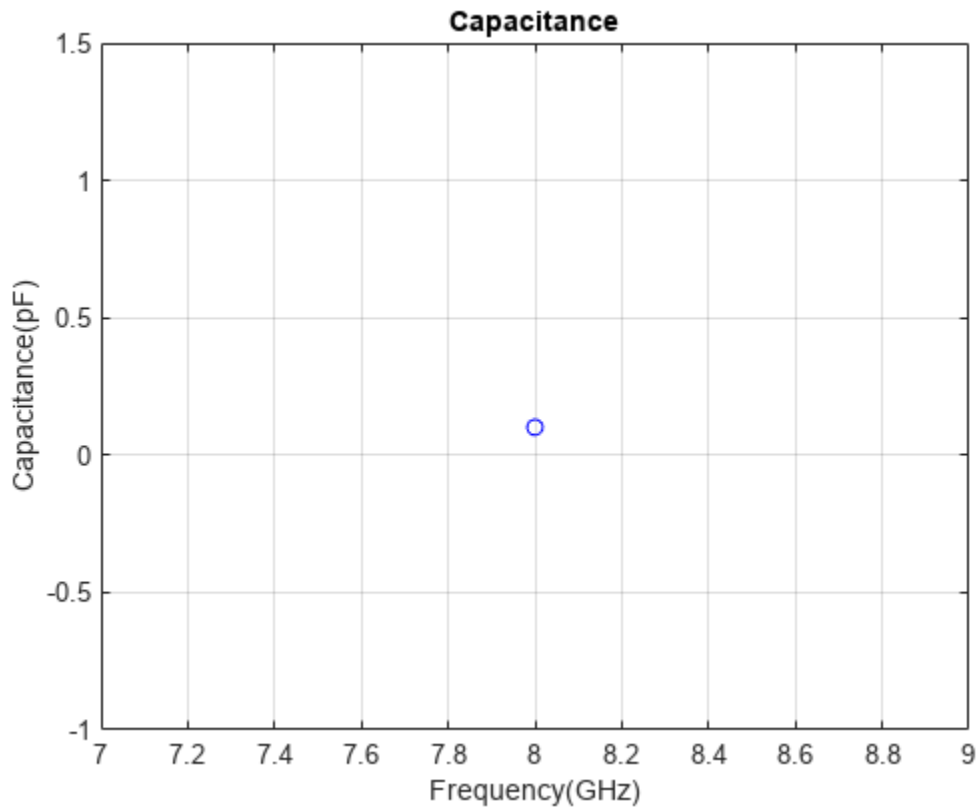
```
capacitor = interdigitalCapacitor

capacitor =
    interdigitalCapacitor with properties:

        NumFingers: 4
        FingerLength: 0.0137
        FingerWidth: 3.1600e-04
        FingerSpacing: 3.0000e-04
        FingerEdgeGap: 3.4100e-04
        TerminalStripWidth: 5.0000e-04
        PortLineWidth: 0.0019
        PortLineLength: 0.0030
        Height: 7.8700e-04
        GroundPlaneWidth: 0.0030
        Substrate: [1x1 dielectric]
        Conductor: [1x1 metal]
```

Calculate the capacitance of the capacitor at 8 GHz.

```
capacitance(capacitor, 8e9, DeEmbed=1, IncludeParasitics=1)
```



Input Arguments

object — Interdigital capacitor

interdigitalCapacitor object

Interdigital capacitor, specified as an interdigitalCapacitor object.

Data Types: char | string

frequency — Frequency to calculate capacitance

nonnegative scalar | vector

Frequency to calculate the capacitance in hertz, specified as a nonnegative scalar or vector of nonnegative elements.

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1, ..., NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: DeEmbed=1

DeEmbed — Deembed feeder line

1 (default) | 0

Deembed feeder line, specified as 0 or 1. When you specify 1, the function deembeds the feeder line.

IncludeParasitics — Include parasitic effects

1 (default) | 0

Include parasitic effects, specified as 0 or 1. When you specify 1, the function includes the parasitic effects.

Z0 — Input and output line impedance

50 (default) | positive scalar

Input and output line impedance to feed the capacitor, specified as a positive scalar in ohms.

Output Arguments**c — Capacitance of capacitor**

scalar | vector

Capacitance of the capacitor, returned as a scalar or vector in farads.

Data Types: double

Version History**Introduced in R2021b****See Also**

inductance

qualityfactor

Calculates and plots Q-factor of capacitor

Syntax

```
qualityfactor(objectfrequency)
qf = qualityfactor(objectfrequency)
qualityfactor(object, frequency, Name, Value)
qf = qualityfactor(object, frequency, Name, Value)
```

Description

`qualityfactor(objectfrequency)` calculates and plots the Q-factor (quality factor) of the capacitor over the specified frequency values in the figure window.

`qf = qualityfactor(objectfrequency)` returns the Q-factor of the capacitor over the specified frequency values.

`qualityfactor(object, frequency, Name, Value)` sets properties using one or more name-value pairs.

`qf = qualityfactor(object, frequency, Name, Value)` sets properties using one or more name-value pairs.

Examples

Input Arguments

object — PCB capacitor component

object handle

PCB capacitor component, specified as an object handle.

Example:

Data Types: `char` | `string`

frequency — Frequency to calculate inductance in hertz

nonnegative scalar or vector

Frequency to calculate inductance in hertz, specified as a nonnegative scalar or vector.

Example:

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name in quotes.

Example: 'DeEmbed', 1

DeEmbed — Deembed feeder line

1 (default) | 0

Deembed feeder line when set to 1, specified as 0 or 1.

Z0 — Input and output line impedance in ohms

50 (default) | positive scalar

Input and output line impedance to feed the capacitor in ohms, specified as a positive scalar.

Output Arguments**qf — Quality factor of capacitor**

scalar or vector

Quality factor of the capacitor, returned as a scalar or vector.

Data Types: double

Version History

Introduced in R2021b

See Also

qualityfactor

Calculates and plots Q-factor of inductor

Syntax

```
qualityfactor(objectfrequency)  
qf = qualityfactor(objectfrequency)
```

Description

`qualityfactor(objectfrequency)` calculates and plots the Q-factor (quality factor) of the inductor over the specified frequency values in the figure window.

`qf = qualityfactor(objectfrequency)` returns the Q-factor of the inductor over the specified frequency values.

Examples

Input Arguments

object — PCB inductor component

object handle

PCB inductor component, specified as an object handle.

Example:

Data Types: `char` | `string`

frequency — Frequency to calculate inductance in hertz

nonnegative scalar or vector

Frequency to calculate inductance in hertz, specified as a nonnegative scalar or vector.

Example:

Data Types: `double`

Output Arguments

qf — Quality factor of inductor

scalar or vector

Quality factor of the inductor, returned as a scalar or vector.

Data Types: `double`

Version History

Introduced in R2021b

See Also

metal

Conductor material

Syntax

```
m = metal(material)
m = metal(Name=Value)
```

Description

`m = metal(material)` returns the metal used as a conductor in the PCB components. You can specify a material from the `MetalCatalog`. The default value for material is perfect electric conductor (PEC).

`m = metal(Name=Value)` returns the metal based on the properties specified by one or more .name-value pairs.

Examples

Microstrip Line with Copper Conductor

Create a microstrip transmission line with copper conductor.

```
mline = microstripLine;
```

Create a copper metal conductor.

```
m = metal('copper')
```

```
m =
  metal with properties:
      Name: 'Copper'
  Conductivity: 59600000
  Thickness: 3.5560e-05
```

For more materials see catalog

Change the microstrip transmission line conductor to copper.

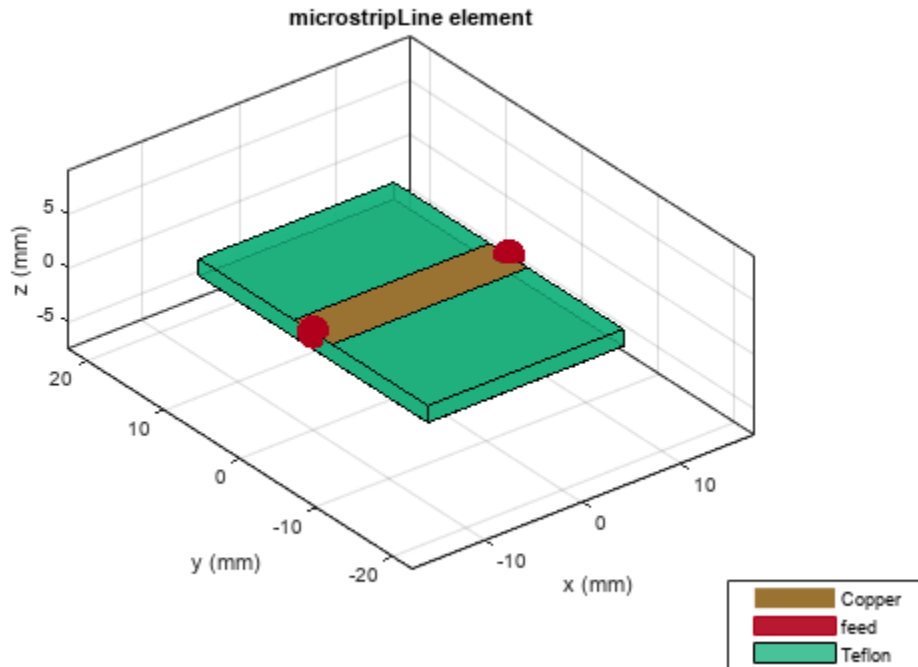
```
mline.Conductor = m

mline =
  microstripLine with properties:
      Length: 0.0200
      Width: 0.0050
      Height: 0.0016
  GroundPlaneWidth: 0.0300
  Substrate: [1x1 dielectric]
```

```
Conductor: [1x1 metal]
```

View the microstrip transmission line.

```
show(mline)
```



Input Arguments

material – Material from metal catalog

'PEC' (default) | character vector

Material from the dielectric catalog, specified as a metal from the `MetalCatalog`. The default material is perfect electric conductor (PEC), which has infinite conductivity and zero thickness.

Example: 'Iron'

Data Types: char

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Name='Iron'`

Name — Name of metal material

'PEC' (default) | character vector

Name of the metal material you want to use as a conductor, specified as a character vector.

Example: Name='Iron'

Data Types: char

Conductivity — Conductivity of metal material

Inf (default) | scalar

Conductivity of the metal material, specified as a scalar in Siemens per meter(S/m). If you set 'Conductivity' to 'Inf', you must set 'Thickness' to '0'.

Example: Conductivity=4.8e06

Data Types: double

Thickness — Thickness of metal

0 (default) | scalar

Thickness of the metal material along the default z-axis, specified as a scalar in meters.

Example: Thickness=0.26e-6

Data Types: double

Output Arguments**m — Conductor metal**

metal object

Conductor metal, returned as a metal object.

Version History

Introduced in R2021a

See Also

MetalCatalog

MetalCatalog

Catalog of metals

Syntax

```
mc = MetalCatalog
```

Description

mc = MetalCatalog creates an object handle for the metal catalog.

- To open the metal catalog, use `open(mc)`
- To see the properties of a metal from the metal catalog, use `s = find(mc, name)`.

Examples

Use Metal Catalog to Design Coplanar Waveguide

Open the metal catalog.

```
mc = MetalCatalog;
open(mc)
```

	Name	Conductivity	Thickness	Units	Comments
1	PEC	Inf	0 m		
2	Copper	59.6000e+006	1.4000 mil		1 oz
3	Aluminium	37.7000e+006	30 mil		
4	Gold	41.1000e+006	0.2000 um		
5	Silver	63.0000e+006	0.2000 um		
6	Zinc	16.9000e+006	4 mil		
7	Tungsten	17.9000e+006	0.2000 um		
8	Lead	4.5500e+006	0.2000 um		
9	Iron	10.0000e+006	0.2000 um		
10	Steel	6.9900e+006	0.6800 mm		
11	Brass	15.0000e+006	0.6000 mm		

List the properties of the metal material Aluminium.

```
s = find(mc, 'Aluminium')
s = struct with fields:
    Name: 'Aluminium'
    Conductivity: 37700000
    Thickness: 30
    Units: 'mil'
    Comments: ''
```

Use the above metal in a coplanar waveguide.


```

m = metal('Aluminium');
waveguide = coplanarWaveguide('Conductor',m)

waveguide =
    coplanarWaveguide with properties:

        Length: 0.0231
        Width: 0.0039
        Spacing: 2.0000e-04
        ViaSpacing: [0.0011 0.0070]
        ViaDiameter: 5.0000e-04
        Height: 0.0016
        GroundPlaneWidth: 0.0300
        Substrate: [1x1 dielectric]
        Conductor: [1x1 metal]

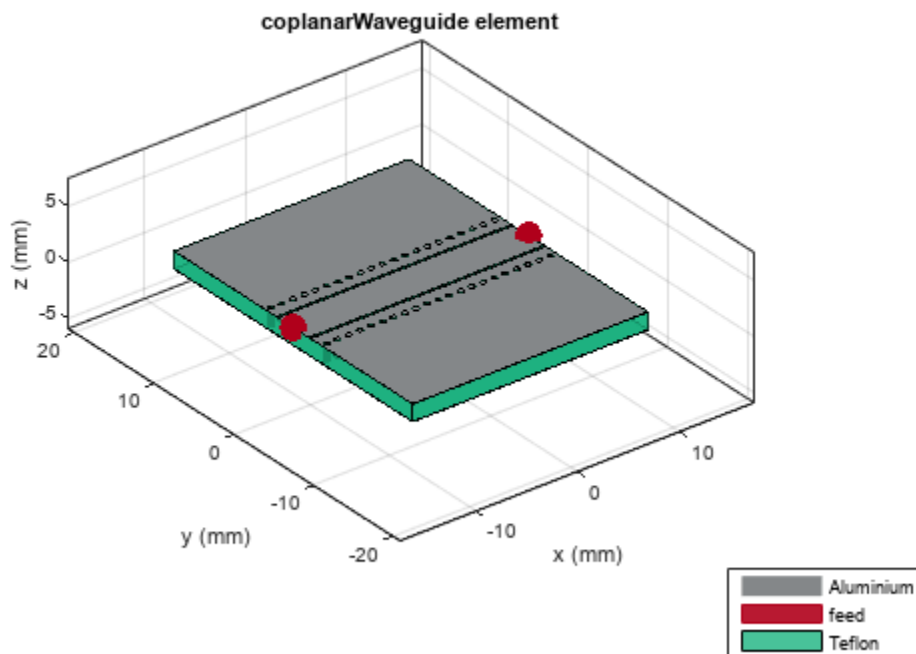
```

View the waveguide using show function.

```

figure;
show(waveguide)

```



Input Arguments

name — Name of metal

'PEC' (default) | character vector

Name of the metal from the metal catalog, specified as a character vector.

Example: 'Copper'

Data Types: char

mc — Metal catalog

metal object

Metal catalog, specified as an object.

Output Arguments**mc — Metal catalog**

object

Metal catalog, returned as an object.

s — Parameters of metal

structure

Parameters of the specified metal from the metal catalog, returned as a structure.

Version History

Introduced in R2021a

See Also

metal

dielectric

Dielectric material for use as substrate

Syntax

```
d = dielectric(material)
d = dielectric(Name=Value)
```

Description

`d = dielectric(material)` returns dielectric materials for use as a substrate in PCB components.

`d = dielectric(Name=Value)` returns dielectric materials based on the properties specified by one or more Name, Value pair arguments.

Examples

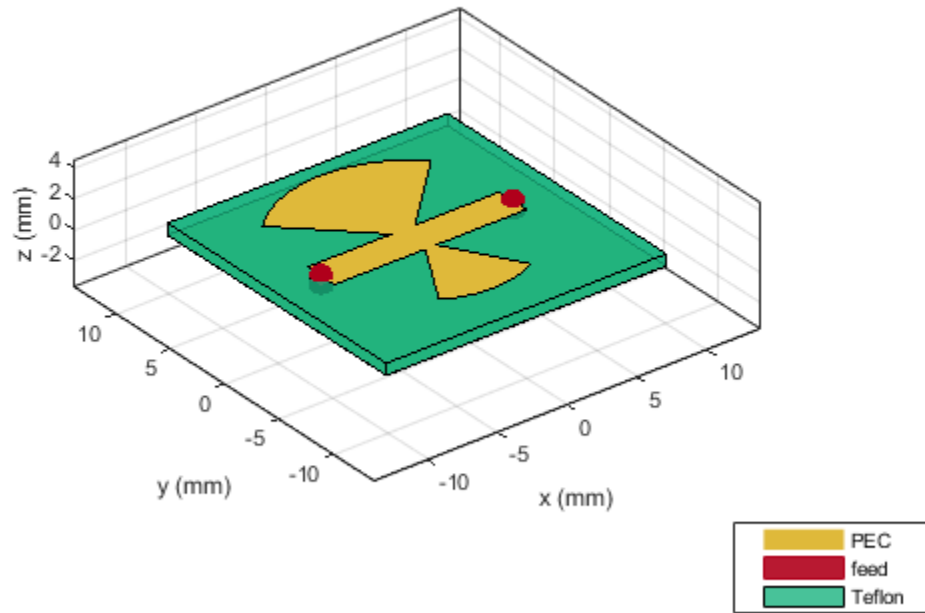
Create Double Shunt Radial Stub

Create shunt radial stub of type double.

```
stub = stubRadialShunt(StubType='double');
stub.OuterRadius = [0.0085 0.0065];
stub.InnerRadius = [0.0012 0.0008];
stub.Angle       = [90 60];
```

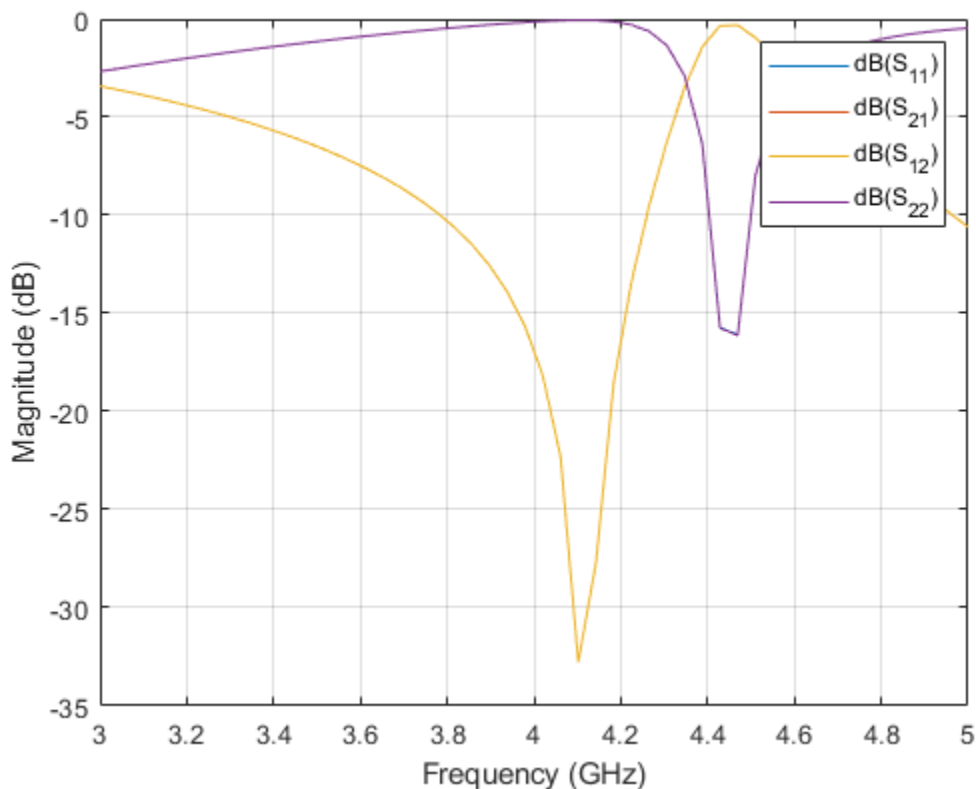
View shunt radial stub.

```
show(stub)
```



Plot s-parameters.

```
spar = sparameters(stub, linspace(3e9, 5e9, 50));  
rfplot(spar)
```



Input Arguments

material — Material from dielectric catalog

'Air' (default) | character

Material from the dielectric catalog, specified as one of the values from the `DielectricCatalog`.

Example: 'FR4'

Data Types: char

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Name=Air`

Name — Name of dielectric material

character vector

Name of the dielectric material you want to specify in the output, specified as a character vector.

Example: `Name='Taconic_TLC'`

Data Types: char

EpsilonR — Relative permittivity of dielectric material

1 | real-valued vector

Relative permittivity of the dielectric material, specified as a real-valued vector.

Example: EpsilonR=4.8000

Data Types: double

LossTangent — Loss in dielectric material

0 (default) | real-valued vector

Loss in the dielectric material, specified as a vector.

Example: LossTangent=0.0260

Data Types: double

Thickness — Thickness of dielectric material

0.0060 (default) | positive vector

Thickness of the dielectric material in meters along default z-axis, specified as a positive vector. This property applies only when you call the function with no output arguments.

Example: Thickness=0.05

Data Types: double

Output Arguments**d — Dielectric material**

object handle

Dielectric material, returned as an object handle. You can use the dielectric material object handle to add dielectric material as a substrate to any PCB component.

Version History**Introduced in R2016a****See Also**

DielectricCatalog

DielectricCatalog

Catalog of dielectric materials

Syntax

```
dc = DielectricCatalog
```

Description

dc = DielectricCatalog creates an object handle for the dielectric catalog.

- To open the dielectric catalog, use `open(dc)`
- To see the properties of a dielectric material from the dielectric catalog, use `s = find(dc, name)`.

Examples

Use Dielectric Catalog to Design Coplanar Waveguide

Open the dielectric catalog.

```
dc = DielectricCatalog;
open(dc)
```

	Name	Relative_Permittivity	Loss_Tangent	Frequency	Comments
1	Air	1	0	1.0000e+009	
2	FR4	4.8000	0.0260	100.0000e+0...	
3	Teflon	2.1000	2.0000e-04	100.0000e+0...	
4	Foam	1.0300	1.5000e-04	50.0000e+006	
5	Polystyrene	2.5500	1.0000e-04	100.0000e+0...	
6	Plexiglas	2.5900	0.0068	10.0000e+009	
7	Fused quartz	3.7800	1.0000e-04	10.0000e+009	
8	E glass	6.2200	0.0023	100.0000e+0...	
9	RO4725JXR	2.5500	0.0022	2.5000e+009	
10	RO4730JXR	3	0.0023	2.5000e+009	
11	TMM2	2.4500	0.0020	10.0000e+009	

List the properties of the dielectric substrate Foam.

```
s = find(dc, 'Foam')
s = struct with fields:
    Name: 'Foam'
    Relative_Permittivity: 1.0300
    Loss_Tangent: 1.5000e-04
    Frequency: 50000000
    Comments: ''
```

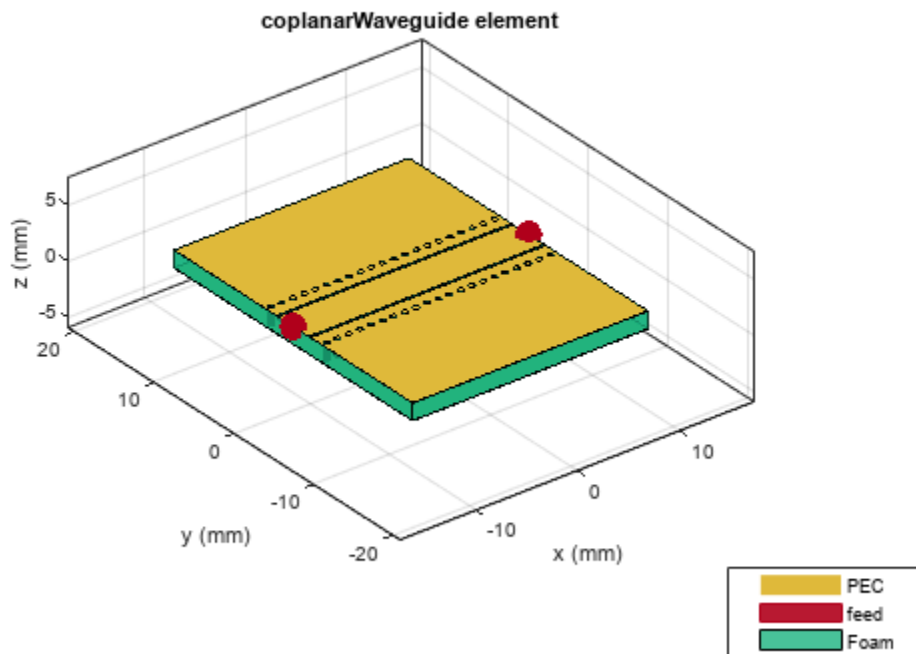
Use the substrate Foam in coplanar waveguide.

```
d = dielectric('Foam');  
waveguide = coplanarWaveguide('Substrate',d)
```

```
waveguide =  
  coplanarWaveguide with properties:  
  
    Length: 0.0231  
    Width: 0.0039  
    Spacing: 2.0000e-04  
    ViaSpacing: [0.0011 0.0070]  
    ViaDiameter: 5.0000e-04  
    Height: 0.0016  
    GroundPlaneWidth: 0.0300  
    Substrate: [1x1 dielectric]  
    Conductor: [1x1 metal]
```

View the waveguide.

```
figure;  
show(waveguide)
```



Input Arguments

name — Name of dielectric material

'Air' (default) | character vector

Name of a dielectric material from the dielectric catalog, specified as a character vector.

Example: 'FR4'

Data Types: char

dc — Dielectric catalog

object handle

Dielectric catalog, specified as an object handle.

Output Arguments

dc — Dielectric catalog

object handle

Dielectric catalog, returned as an object handle.

s — Parameters of dielectric material

structure

Parameters of a dielectric material from the dielectric catalog, returned as a structure.

Version History

Introduced in R2016a

See Also

dielectric

gerberRead

Create PCBReader object with specified Gerber and drill files

Syntax

```
P = gerberRead(T)
P = gerberRead([],B)
P = gerberRead(T,B)
P = gerberRead(T,B,D)
```

Description

P = gerberRead(T) creates a PCBReader object with the top layer Gerber file specified in T.

Note The PCBReader object reads RS-274X Gerber files. It does not support RS-274D Gerber files.

P = gerberRead([],B) creates a PCBReader object with the bottom layer Gerber file specified in B.

P = gerberRead(T,B) creates a PCBReader object with the specified top and bottom layer Gerber file names.

P = gerberRead(T,B,D) creates a PCBReader object with the specified top and bottom layer Gerber files and the drill file specified in D .

Examples

Import and View Top Layer Gerber File

Use the gerberRead function to import a top layer Gerber file.

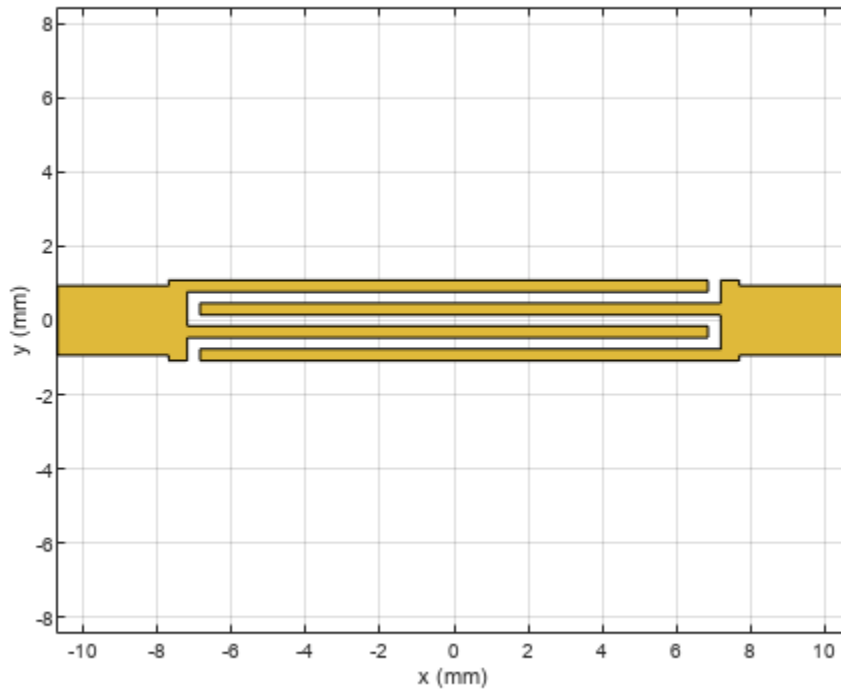
```
P = gerberRead('interdigital_Capacitor.gtl');
```

Extract the metal layer from the file using the shapes function.

```
s = shapes(P);
```

View the top metal layer.

```
show(s)
```



Input Arguments

T — Top layer Gerber file

string scalar | character vector

Top layer Gerber file, specified as a character vector or string scalar. The file must be saved as a GTL file.

Example: `gerberRead('Filetop.gtl');`

B — Bottom layer Gerber file

string scalar | character vector

Bottom layer Gerber file, specified as a character vector or string scalar. The file must be saved as a GBL file.

Example: `gerberRead([], 'FileBottom.gbl');`

D — Drill file

string scalar | character vector

Drill file, specified a character vector or string scalar. You can specify either a DRL or a TXT file.

Example: `gerberRead('Filetop.gtl', 'FileBottom.gbl', 'FileDrill.txt');`

Output Arguments

P – Files read

PCBReader object

Gerber and drill files read, returned as a PCBReader object.

Limitations

Limitations of the gerberRead function while reading a gtl or gbl file are:

Command	Description	Comments
AM	Aperture Macro	Not supported
AB	Aperture Block	Not supported
SR	Step and Repeat	Not supported
TF	File Attributes	Command is ignored and no error is thrown
TA	Aperture Attributes	Command is ignored and no error is thrown
TO	Object Attributes	Command is ignored and no error is thrown
TD	Delete Attributes	Command is ignored and no error is thrown

Cut-ins are not supported.

Version History

Introduced in R2021b

See Also

PCBReader | PCBServices | PCBConnectors | PCBWriter

gerberWrite

Generate Gerber files

Syntax

```
gerberWrite(designObject)
gerberWrite(designObject,writer)
gerberWrite(designObject,writer,rfConnector)
[a,g] = gerberWrite(designObject,writer,rfConnector)
```

Description

`gerberWrite(designObject)` creates Gerber-format files based on multilayer 2.5D design from PCB component stack up.

`gerberWrite(designObject,writer)` creates a Gerber-format files based on multilayer 2.5D design from PCB component using specified PCB writer services.

`gerberWrite(designObject,writer,rfConnector)` creates Gerber-format files based on multilayer 2.5D design from PCB component using a writer object and an RF connector object.

`[a,g] = gerberWrite(designObject,writer,rfConnector)` returns the PCBWriter object, a and the path to the location of the Gerber files.

Examples

Generate Gerber Format Files for Default PCB Component

Create a default PCB component.

```
p = pcbComponent;
```

Use 2 Cinch SMA connectors and the Mayhew Labs PCB viewer.

```
W = PCBServices.MayhewWriter;
C1 = PCBConnectors.SMA_Cinch;
C2 = PCBConnectors.SMA_Cinch;
```

Generate the Gerber-format files.

```
[A,g] = gerberWrite(p,W,{C1,C2})
```

A =

PCBWriter with properties:

```

                Design: [1x1 struct]
                Writer: [1x1 PCBServices.MayhewWriter]
            Connector: {1x2 cell}
    UseDefaultConnector: 0
ComponentBoundaryLineWidth: 8
    ComponentNameFontSize: []
```

```
DesignInfoFontSize: []  
    Font: 'Arial'  
    PCBMargin: 5.0000e-04  
    Soldermask: 'both'  
    Solderpaste: 1
```

See info for details

```
g =  
'C:\TEMP\Bdoc22b_2054784_6060\ibB18F8B\7\tpc19835a0\rfpcb-ex06685827\untitled'
```

Input Arguments

designObject — PCB design geometry file

PCBWriter object

PCB design geometry file, specified as a PCBWriter object.

Example: `a = PCBWriter(p1)` creates a PCBWriter object, `a.gerberWrite(a)` creates a Gerber file using `a`.

rfConnector — RF connector type

PCBConnectors object

RF connector type, specified as a PCBConnectors object.

Example: `c = PCBConnectors.SMA_Cinch`; `gerberWrite(p1,c)` uses SMA_Cinch RF connector at the feedpoint.

writer — PCB service

PCBServices object

PCB service, specified as a PCBServices object.

Example: `s = PCBServices.MayhewWriter`; `gerberWrite(p1,s)` uses Mayhew Labs PCB service to create and view the PCB design.

Output Arguments

a — PCB writer

PCBWriter object

PCB writer that generated the Gerber files, returned as a PCBWriter object.

g — Path to generated Gerber files folder

character vector

Path to generated Gerber files folder, returned as a character vector.

Version History

Introduced in R2021b

See Also

[PCBServices](#) | [PCBConnectors](#) | [PCBWriter](#)

add

Boolean unite operation on two RF PCB shapes

Syntax

```
c = add(shape1, shape2)
```

Description

`c = add(shape1, shape2)` unites `shape1` and `shape2` using the add operation. You can also use the `+` symbol to add the two shapes together.

Examples

Add Two RF PCB Shapes

Create a curved bend shape with a length of 5 mm.

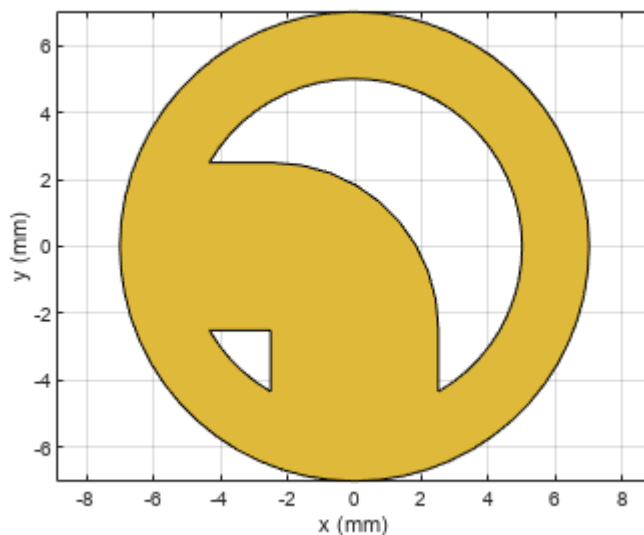
```
bend = bendCurved(Length=[5e-3 5e-3]);
```

Create an annular ring shape with the default inner radius of 5 m.

```
ring = ringAnnular;
```

Add the two shapes and display the result.

```
shapeSum = add(bend, ring);  
show(shapeSum)
```



Add Two RF PCB Shapes Using + Operator

Create the default curve shape.

```
shape1 = curve;
```

Create a right angle U-bend shape with an adjusted size and position to complement the curve shape.

```
shape2 = ubendRightAngle;
```

Add the two shapes using the + operator, and display the resulting Polygon object.

```
shapeSum = shape1+shape2
```

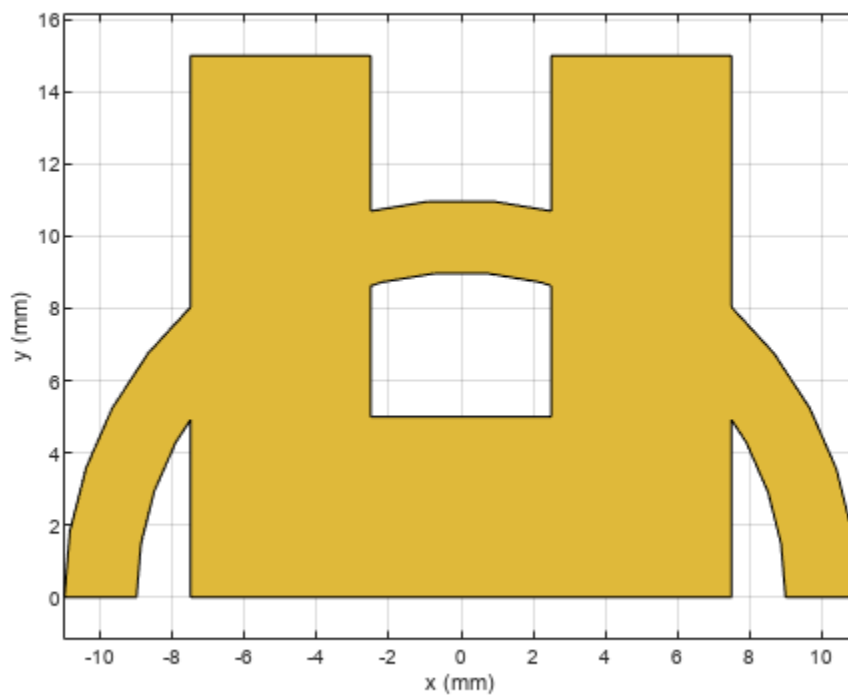
```
shapeSum =
```

```
  Polygon with properties:
```

```
    Name: 'mypolygon'
```

```
    Vertices: [43x3 double]
```

```
show(shapeSum)
```



Input Arguments

shape1 — First shape

object

First shape created using custom elements and shape objects of RF PCB Toolbox™, specified as an object.

Example: `shape1 = bendCurved`; specifies the first shape as a `bendCurved` object.

shape2 — Second shape

object

Second shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape2 = ringAnnular`; specifies the second shape as a `ringAnnular` object.

Version History

Introduced in R2021b

See Also

`area` | `intersect` | `subtract` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot` | `scale`

and

Shape1 & Shape2 for RF PCB shapes

Syntax

```
c = and(shape1, shape2)
```

Description

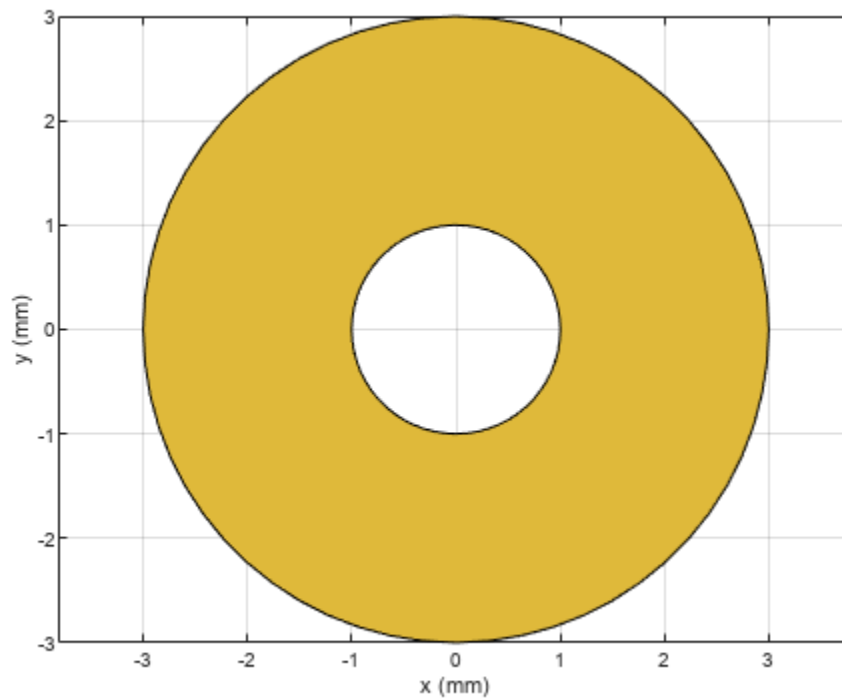
`c = and(shape1, shape2)` calls the syntax `shape1` & `shape2` to intersect two shapes.

Examples

Intersect Two RF PCB Shapes

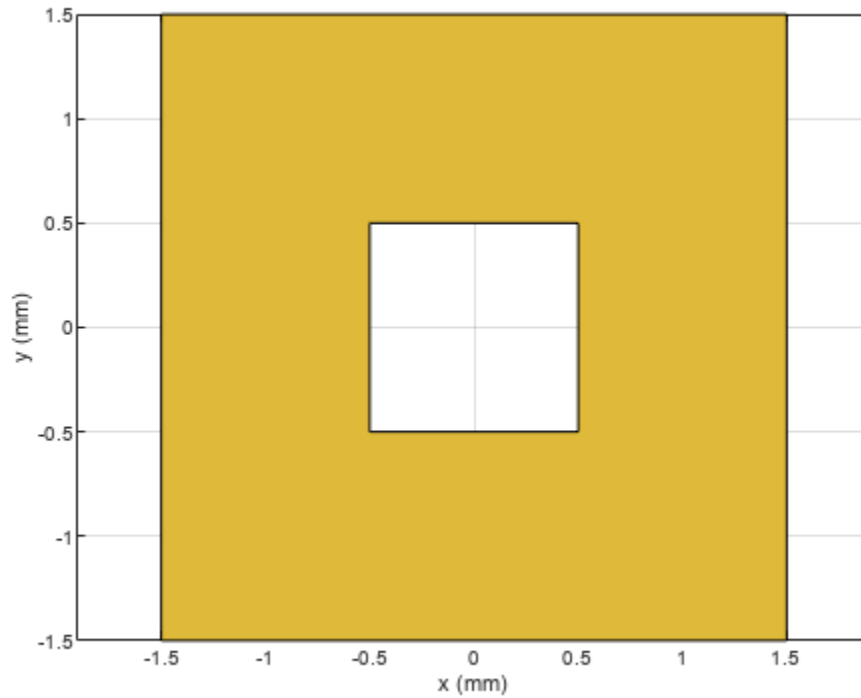
Create and display an annular ring.

```
shape1 = ringAnnular(InnerRadius=1e-3);  
show(shape1)
```



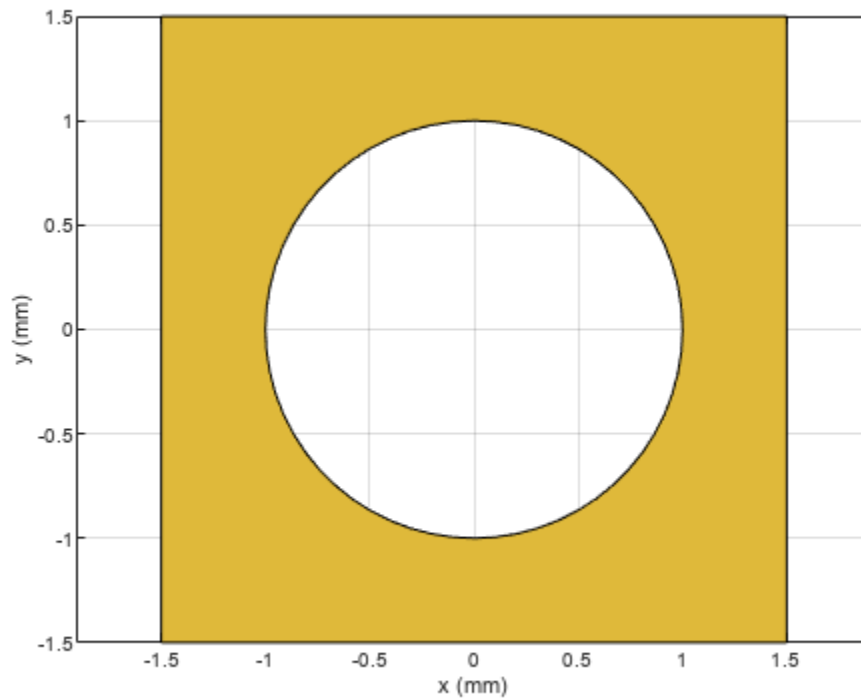
Create and display a square ring.

```
shape2 = ringSquare(InnerSide=1e-3);  
show(shape2)
```



Find and display the intersection of the shapes.

```
shapeIntersection = and(shape1,shape2);  
show(shapeIntersection)
```



Intersect Two RF PCB Shapes Using & Operator

Create and display an annular ring.

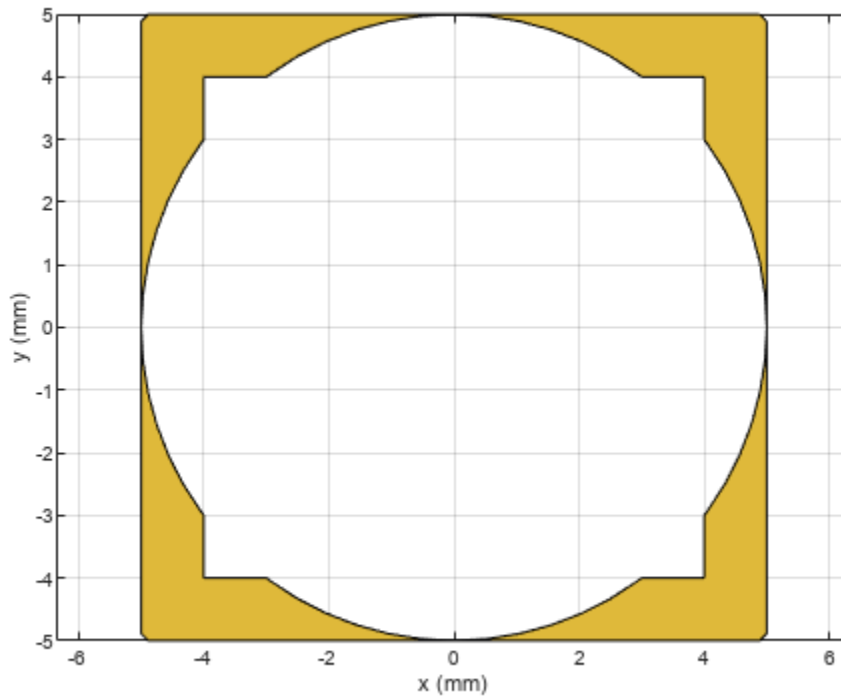
```
shape1 = ringAnnular;
```

Create and display a square ring.

```
shape2 = ringSquare('InnerSide',8e-3);
```

Find and display the intersection of the shapes.

```
shapeIntersection = shape1&shape2;  
show(shapeIntersection)
```



Input Arguments

shape1 — First shape
object

First shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape1 = bendCurved`; specifies the first shape as a `bendCurved` object.

shape2 — Second shape
object

Second shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape2 = ringAnnular`; specifies the second shape as a `ringAnnular` object.

Version History

Introduced in R2021b

See Also

add | subtract | area | intersect | rotate | rotateX | rotateY | rotateZ | translate | show
| mesh | plot | scale

area

Calculate area of RF PCB shape in square meters

Syntax

```
a = area(shape)
```

Description

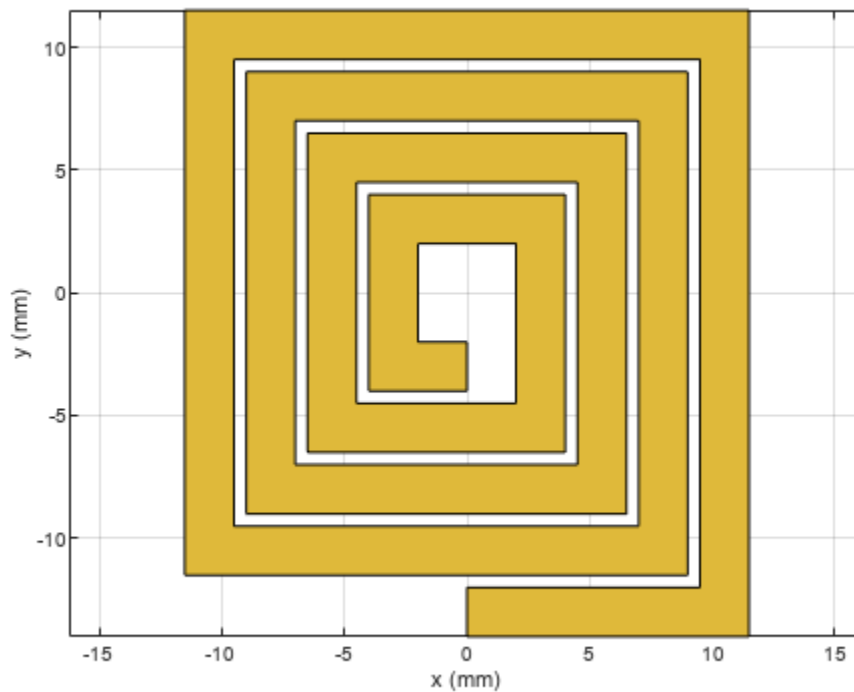
`a = area(shape)` calculates the area of the shape in units of square meters.

Examples

Calculate Area of Spiral Trace

Create and view a default spiral trace.

```
trace = traceSpiral;  
show(trace)
```



Get the area of the spiral trace.


```
a = area(trace)
```

```
a = 4.5200e-04
```

Input Arguments

shape — RF PCB shape

object

RF PCB shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape = bendCurved`; specifies the shape as a `bendCurved` object.

Version History

Introduced in R2021b

See Also

`add` | `subtract` | `intersect` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot` | `scale`

intersect

Boolean intersection operation on two RF PCB shapes

Syntax

```
c = intersect(shape1,shape2)
```

Description

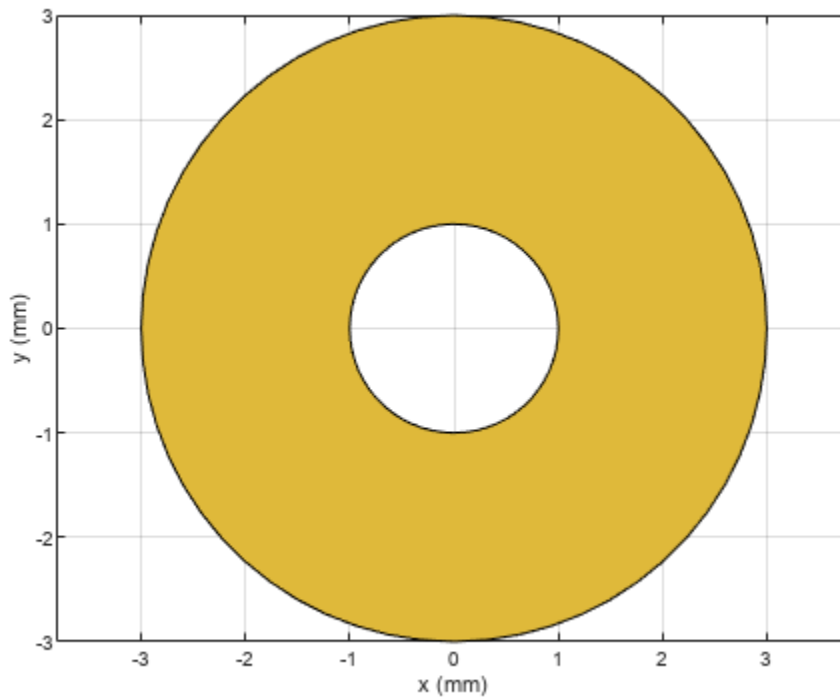
`c = intersect(shape1,shape2)` intersects `shape1` and `shape2` using the intersect operation. You can also use the `&` to intersect the two shapes.

Examples

Boolean Intersection of Two RF PCB Shapes

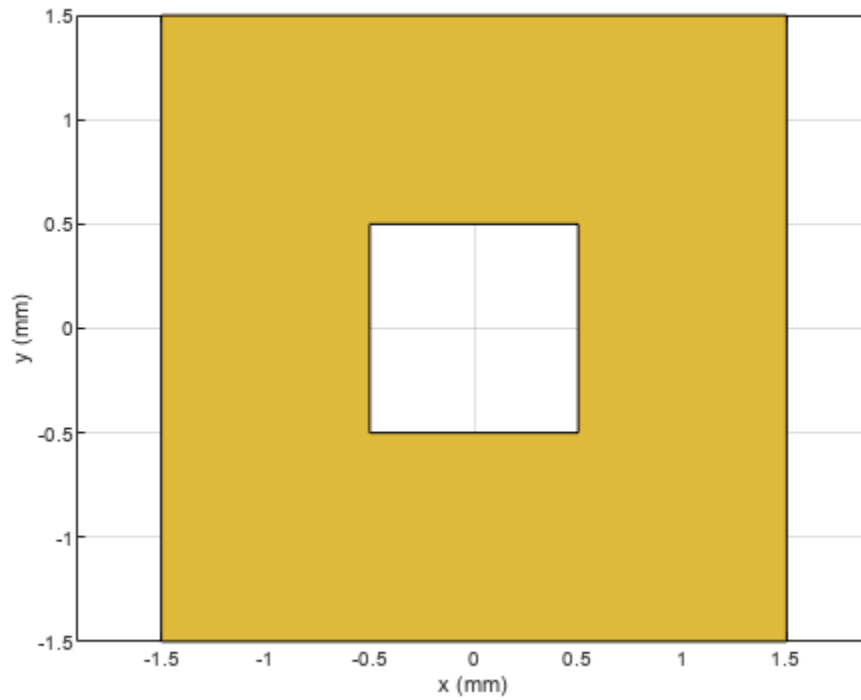
Create and display an annular ring.

```
shape1 = ringAnnular(InnerRadius=1e-3);  
show(shape1)
```



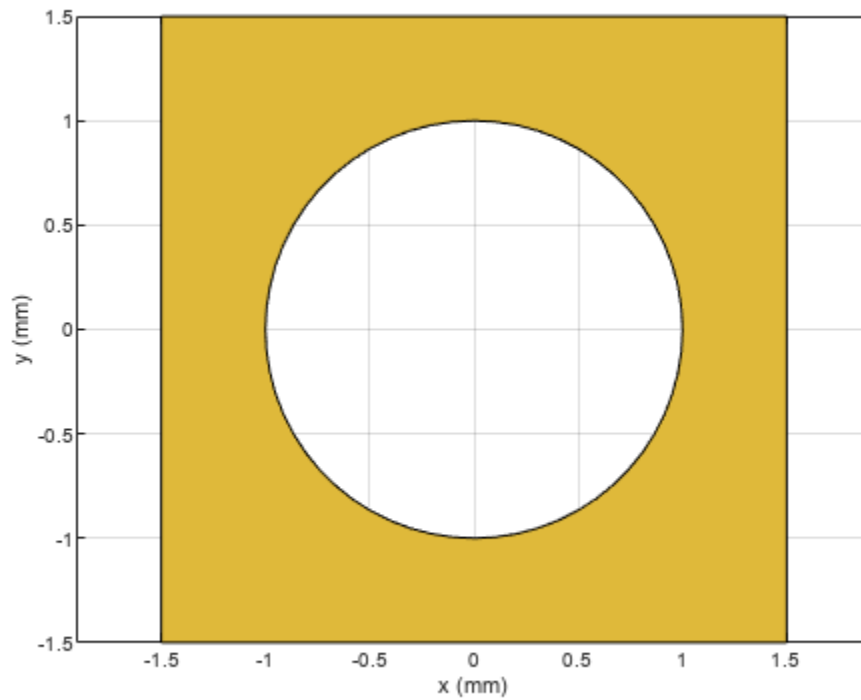
Create and display a square ring.

```
shape2 = ringSquare(InnerSide=1e-3);  
show(shape2)
```



Find and display the intersection of the shapes.

```
shapeIntersection = intersect(shape1,shape2);  
show(shapeIntersection)
```



Intersect Two RF PCB Shapes Using & Operator

Create and display an annular ring.

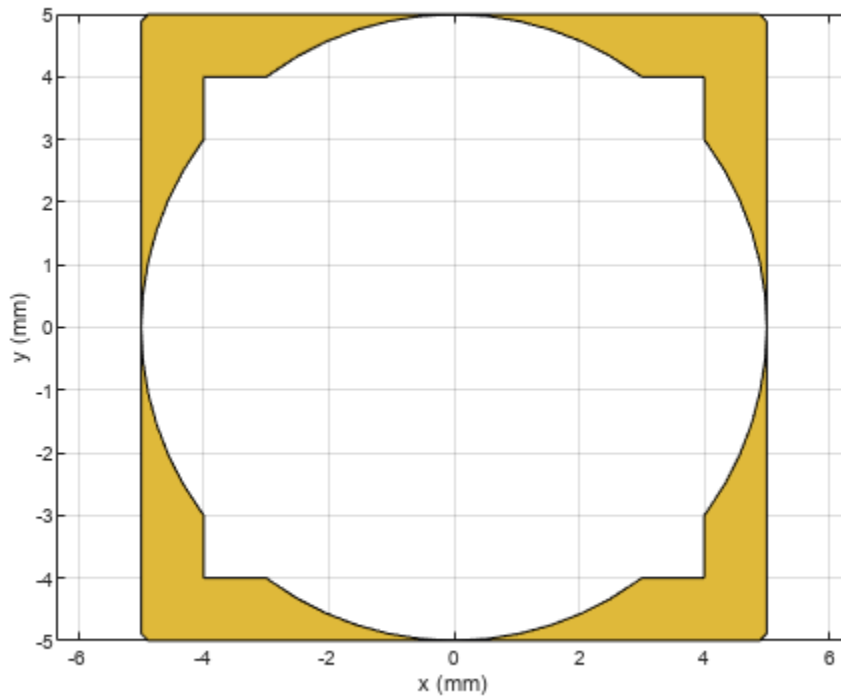
```
shape1 = ringAnnular;
```

Create and display a square ring.

```
shape2 = ringSquare('InnerSide',8e-3);
```

Find and display the intersection of the shapes.

```
shapeIntersection = shape1&shape2;  
show(shapeIntersection)
```



Input Arguments

shape1 — First shape
object

First shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape1 = bendCurved`; specifies the first shape as a `bendCurved` object.

shape2 — Second shape
object

Second shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape2 = ringAnnular`; specifies the second shape as a `ringAnnular` object.

Version History

Introduced in R2021b

See Also

add | subtract | area | rotate | rotateX | rotateY | rotateZ | translate | show | mesh | plot

minus

Shape1 - Shape2 for RF PCB shapes

Syntax

```
c = minus(shape1,shape2)
```

Description

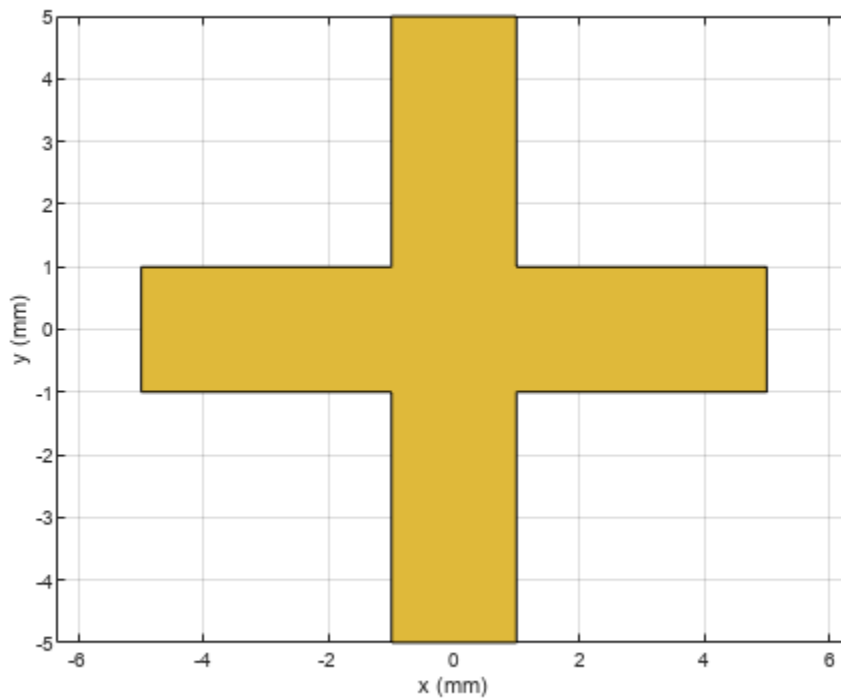
`c = minus(shape1,shape2)` calls the syntax `shape1 - shape2` to subtract two shapes.

Examples

Boolean Subtraction of Two RF PCB Shapes

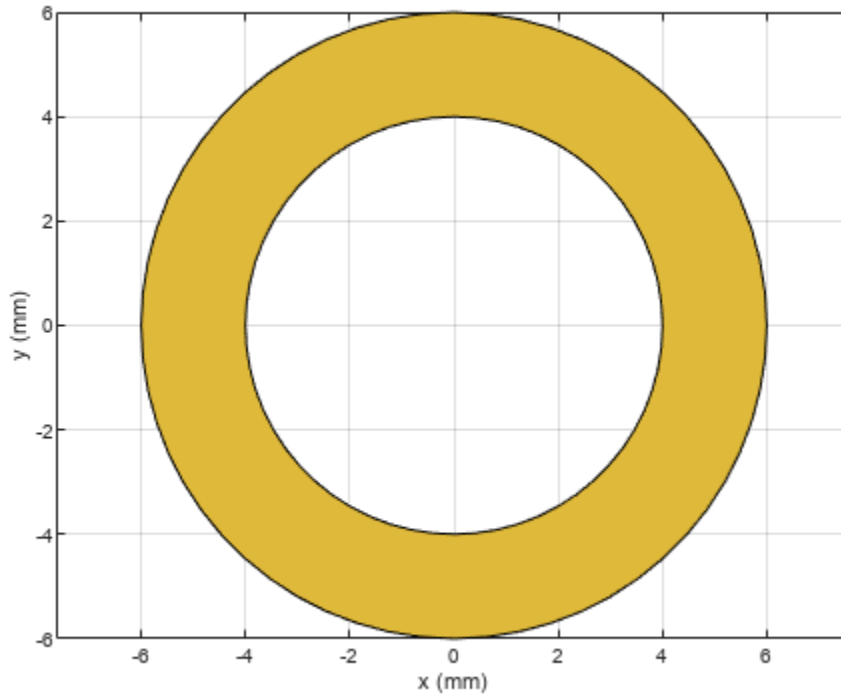
Create and display a cross trace shape.

```
trace = traceCross;  
show(trace)
```



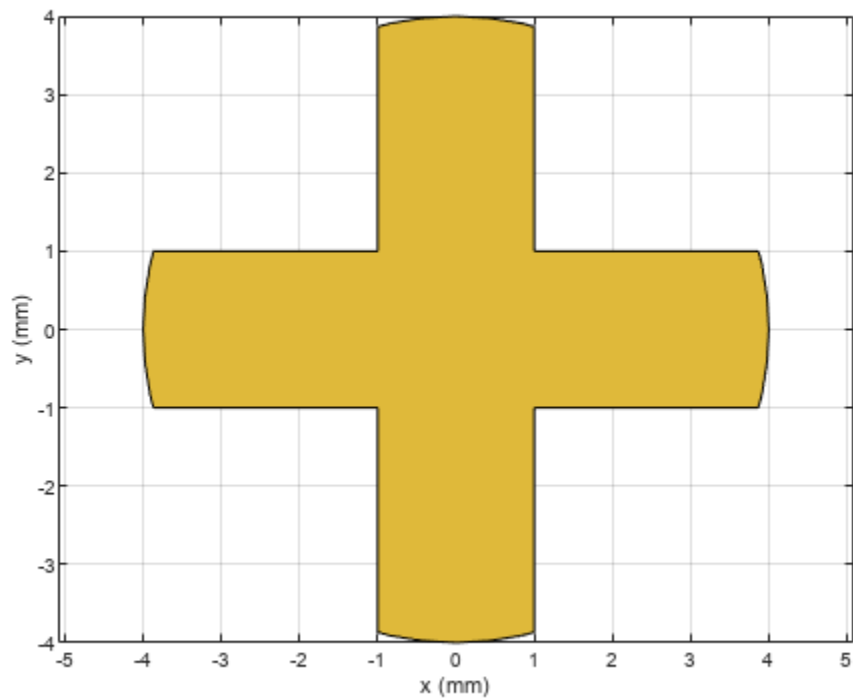
Create and display an annular ring shape with an inner radius of 4 mm.

```
ring = ringAnnular(InnerRadius=4e-3);  
show(ring)
```



Subtract the annular ring from the cross trace and display the result.

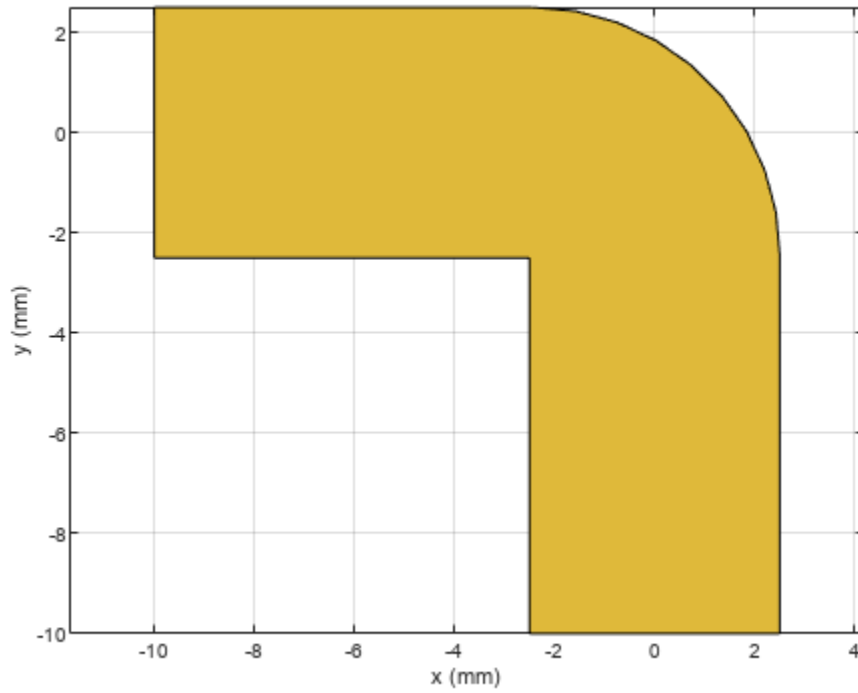
```
shapeDiff = minus(trace,ring);  
show(shapeDiff)
```

Subtract Two RF PCB Shapes Using - Operator

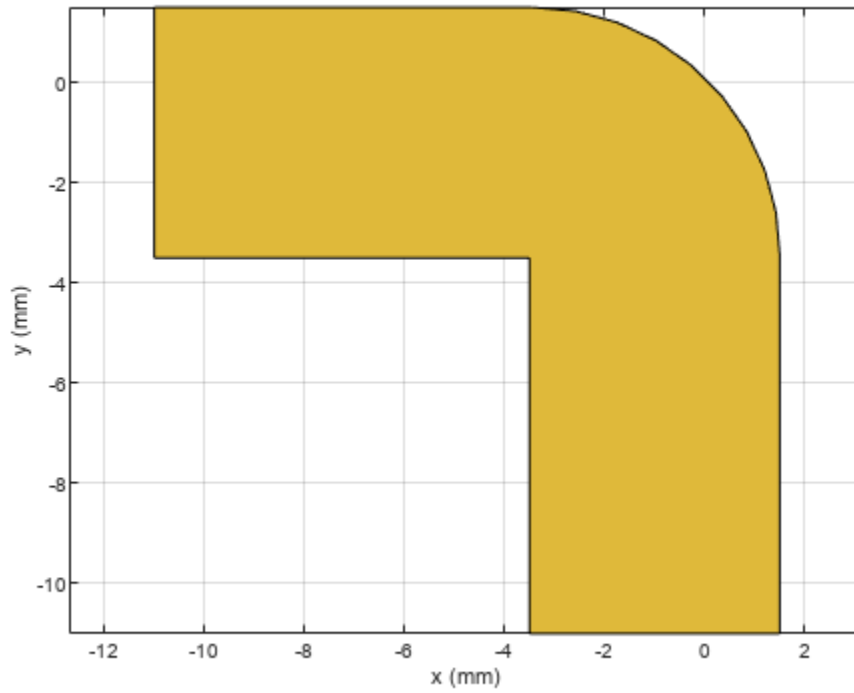
Create and display a curved bend shape.

```
bend1 = bendCurved;  
show(bend1)
```



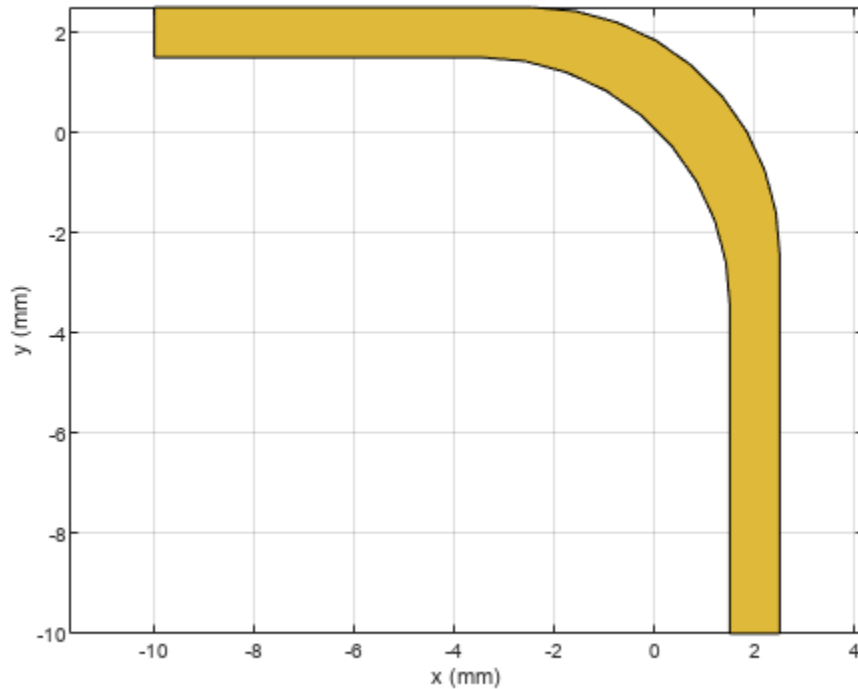
Create and display a curved bend shape with a spatial offset.

```
bend2 = bendCurved(ReferencePoint=[-1e-3 -1e-3]);  
show(bend2)
```



Subtract the offset bend from the default bend and display the result.

```
shapeDiff = bend1 - bend2;  
show(shapeDiff)
```



Input Arguments

shape1 — First shape
object

First shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape1 = bendCurved`; specifies the first shape as a `bendCurved` object.

shape2 — Second shape
object

Second shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape2 = ringAnnular`; specifies the second shape as a `ringAnnular` object.

Version History

Introduced in R2021b

See Also

add | subtract | area | intersect | rotate | rotateX | rotateY | rotateZ | translate | show
| mesh | plot | scale

plot

Plot boundary of RF PCB shape

Syntax

```
plot(shape)  
plot(shape,Name,Value)  
p = plot( ___ )
```

Description

`plot(shape)` plots the boundary of the shape.

`plot(shape,Name,Value)` specifies the line properties using one or more name-value arguments.

Example: `plot(shape,Color="r",LineWidth=2)` plots the boundary of the shape as a red line with a width of 2 pixels.

`p = plot(___)` returns the line object. Use `p` to modify properties of the line after it is created.

Examples

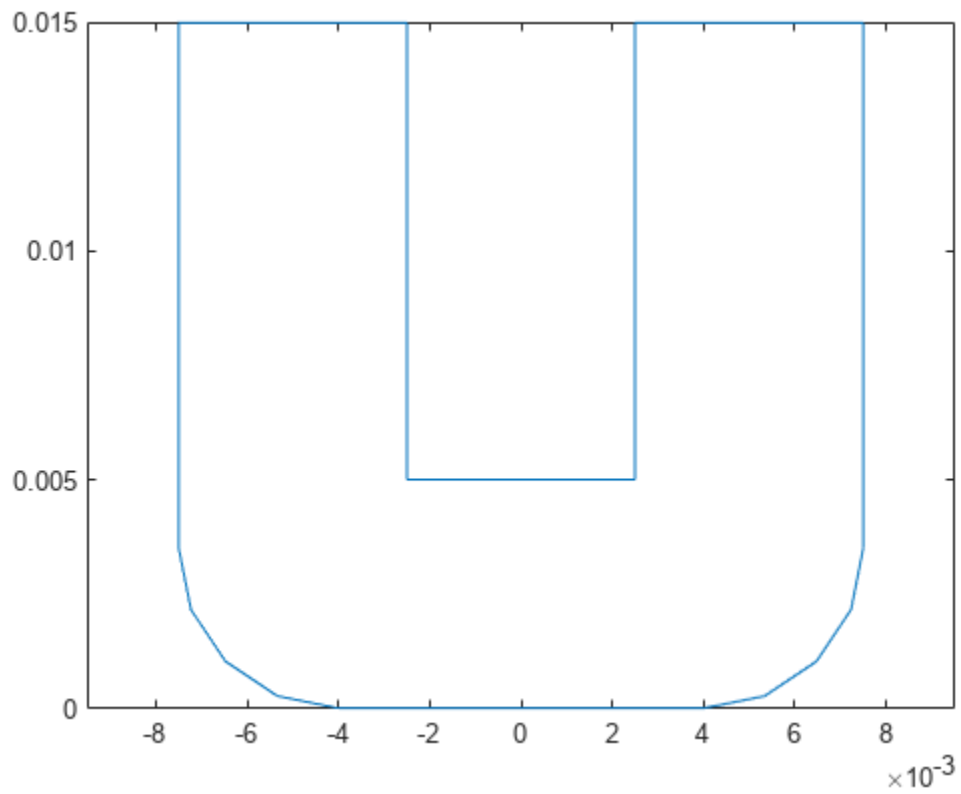
Plot Curved U-Bend

Create a curved U-bend shape.

```
ubend = ubendCurved;
```

Plot the shape.

```
plot(ubend)
```



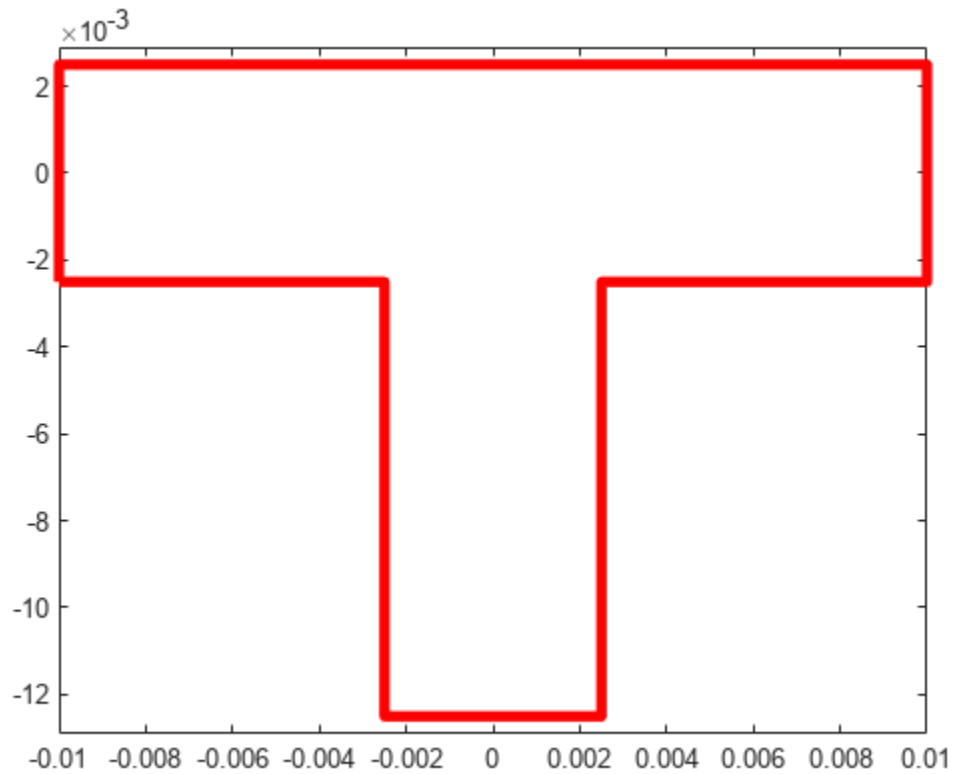
Plot Shape and Modify Line Properties

Create a tee trace shape.

```
trace = traceTee;
```

Plot the shape using a red line of width 4 pixels.

```
plot(trace, LineWidth=4, Color="r")
```



Input Arguments

shape — RF PCB shape
object

RF PCB shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape = bendCurved`; specifies the shape as a `bendCurved` object.

Version History

Introduced in R2021b

See Also

`show` | `mesh`

plus

Shape1 + Shape2 for RF PCB shapes

Syntax

```
c = plus(shape1,shape2)
```

Description

`c = plus(shape1,shape2)` calls the syntax `shape1 + shape2` to unite two shapes.

Examples

Boolean Unite of Two RF PCB Shapes

Create a curved bend shape with a length of 5 m

```
bend = bendCurved(Length=[5 5]);
```

Create an annular ring shape with the default inner radius of 5 m.

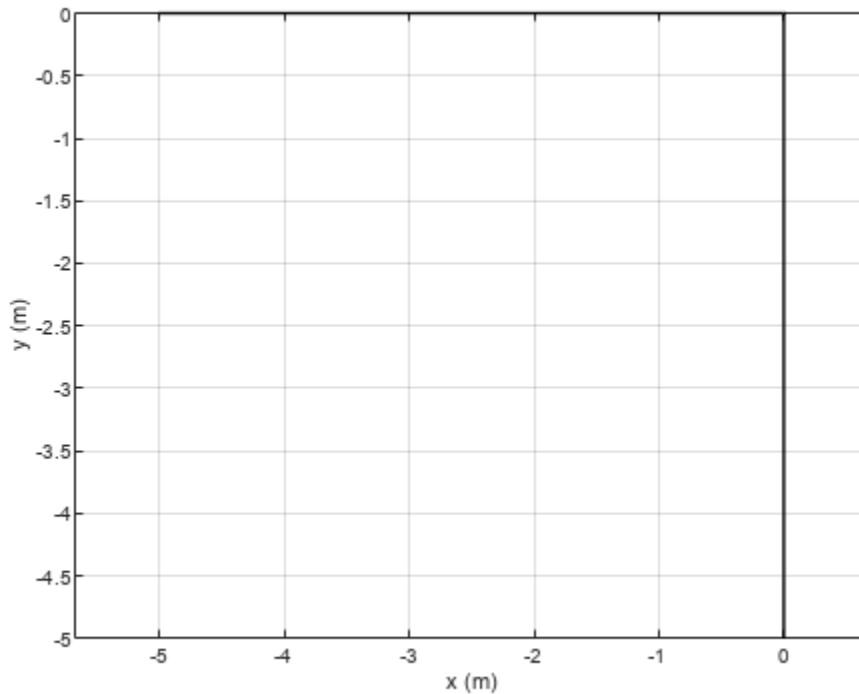
```
ring = ringAnnular;
```

Add the two shapes and display the result.

```
shapeSum = plus(bend,ring)
```

```
shapeSum =  
  Polygon with properties:  
      Name: 'mypolygon'  
      Vertices: [111x3 double]
```

```
show(shapeSum)
```



Add Two RF PCB Shapes Using + Operator

Create the default curve shape.

```
shape1 = curve;
```

Create a right angle U-bend shape with an adjusted size and position to complement the curve shape.

```
shape2 = ubendRightAngle;
```

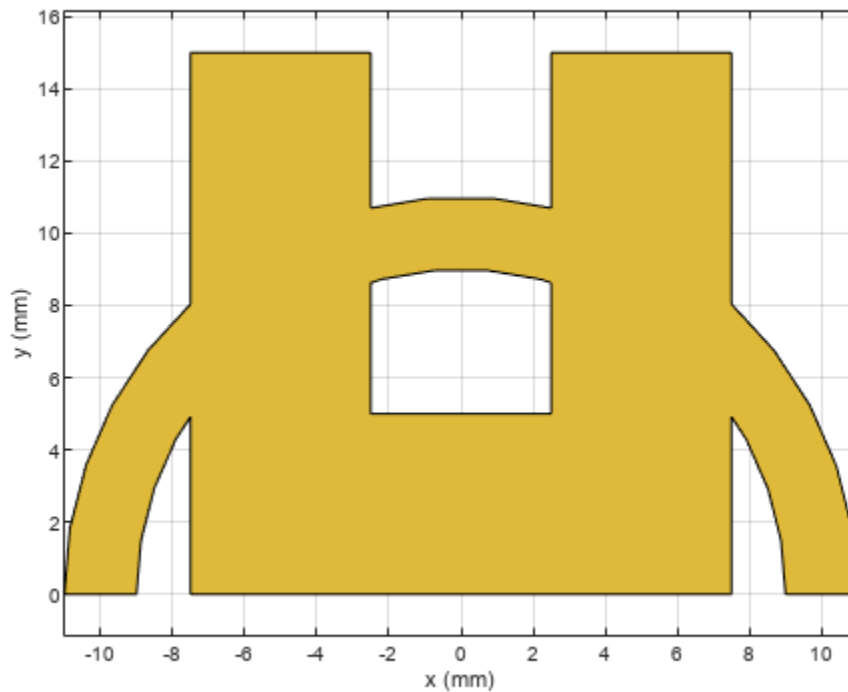
Add the two shapes using the + operator, and display the resulting Polygon object.

```
shapeSum = shape1+shape2
```

```
shapeSum =  
  Polygon with properties:
```

```
    Name: 'mypolygon'  
  Vertices: [43x3 double]
```

```
show(shapeSum)
```



Input Arguments

shape1 — First shape
object

First shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape1 = bendCurved`; specifies the first shape as a `bendCurved` object.

shape2 — Second shape
object

Second shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape2 = ringAnnular`; specifies the second shape as a `ringAnnular` object.

Version History

Introduced in R2021b

See Also

add | subtract | area | intersect | rotate | rotateX | rotateY | rotateZ | translate | show
| mesh | plot | scale

rotate

Rotate RF PCB shape about defined axis

Syntax

```
c = rotate(shape,angle,axis1,axis2)
```

Description

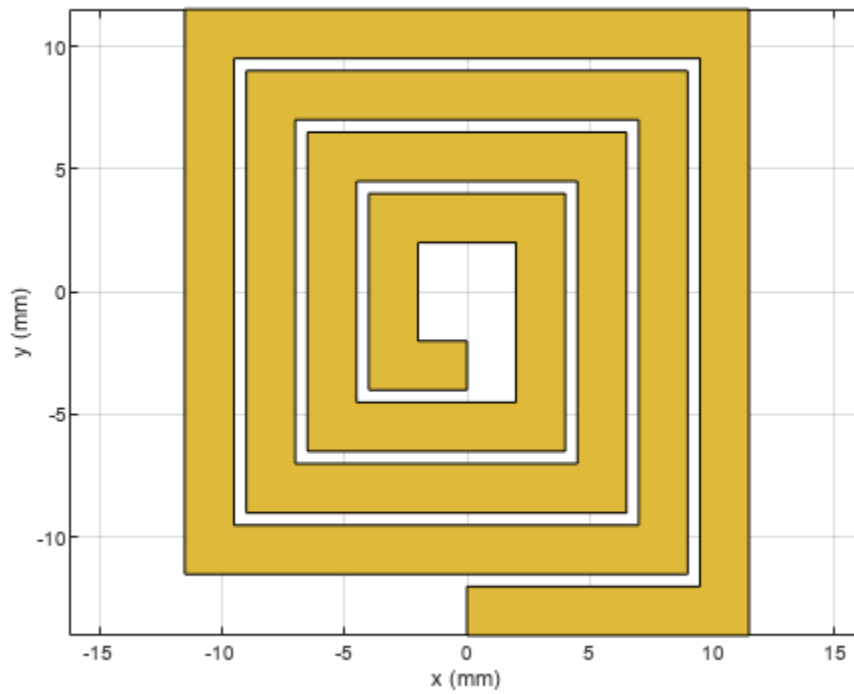
`c = rotate(shape,angle,axis1,axis2)` rotates a shape by a specified angle about an axes defined by two points `axis1` and `axis2`.

Examples

Rotate Spiral Trace About Axis

Create and display a spiral trace.

```
trace = traceSpiral;  
show(trace)
```



Specify two points that define the axes of rotation.

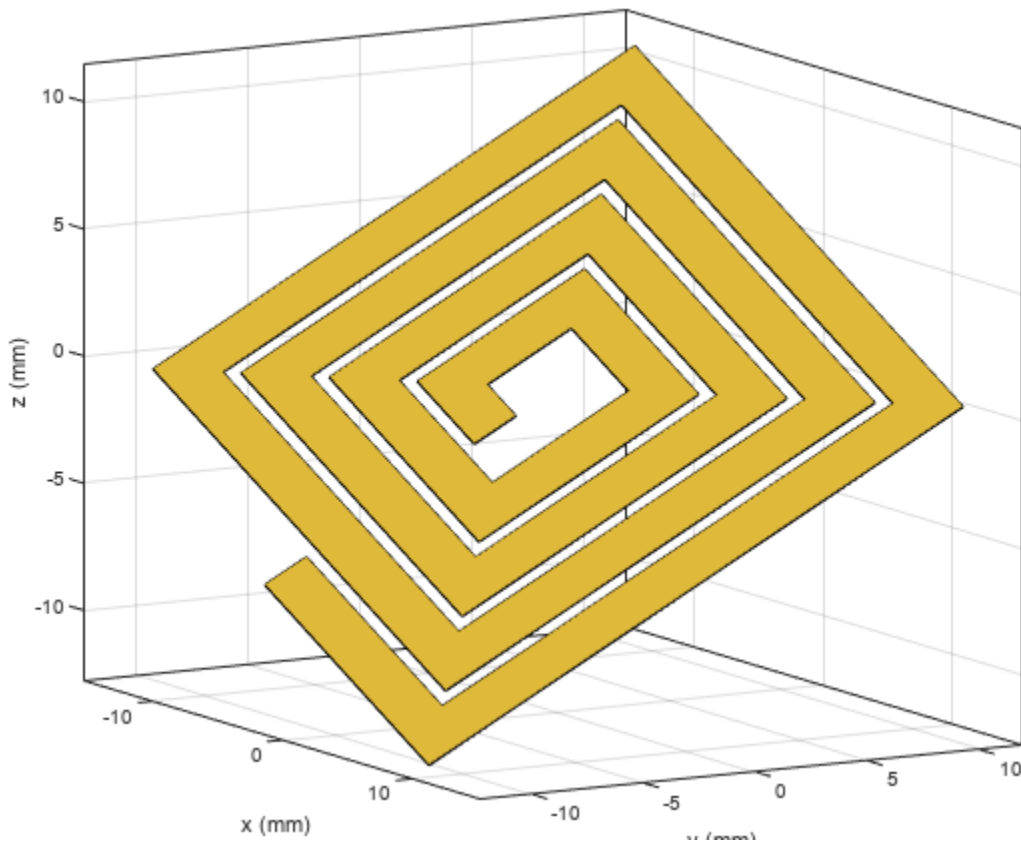
```
pt1 = [0 0 0];  
pt2 = [1 1 0];
```

Rotate the spiral trace by 45 degrees about the axis.

```
traceRot = rotate(trace,45,pt1,pt2);
```

Display the rotated shape. Set the camera line of sight to display in 3-D space.

```
show(traceRot)  
view(60,10)
```



Input Arguments

shape — RF PCB shape

object

RF PCB shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape = bendCurved`; specifies the shape as a `bendCurved` object.

angle — Angle of rotation

scalar

Angle of rotation, specified as a scalar in degrees.

Example: 45 rotates the shape around the axis by 45 degrees.

Data Types: double

axis1 — One point on axis of rotation

three-element vector

One point on the axis of rotation, specified as a three-element vector of Cartesian coordinates in meters.

Example: [0 0 0]

Data Types: double

axis2 — Second point on axis of rotation

three-element vector

Second point on the axis of rotation, specified as a three-element vectors of Cartesian coordinates in meters. **axis2** must be different than **axis1**.

Example: [0 0 1]

Data Types: double

Version History

Introduced in R2021b

See Also

add | subtract | area | intersect | rotateX | rotateY | rotateZ | translate | show | mesh | plot | scale

rotateX

Rotate RF PCB shape about x-axis

Syntax

```
c = rotateX(shape,angle)
```

Description

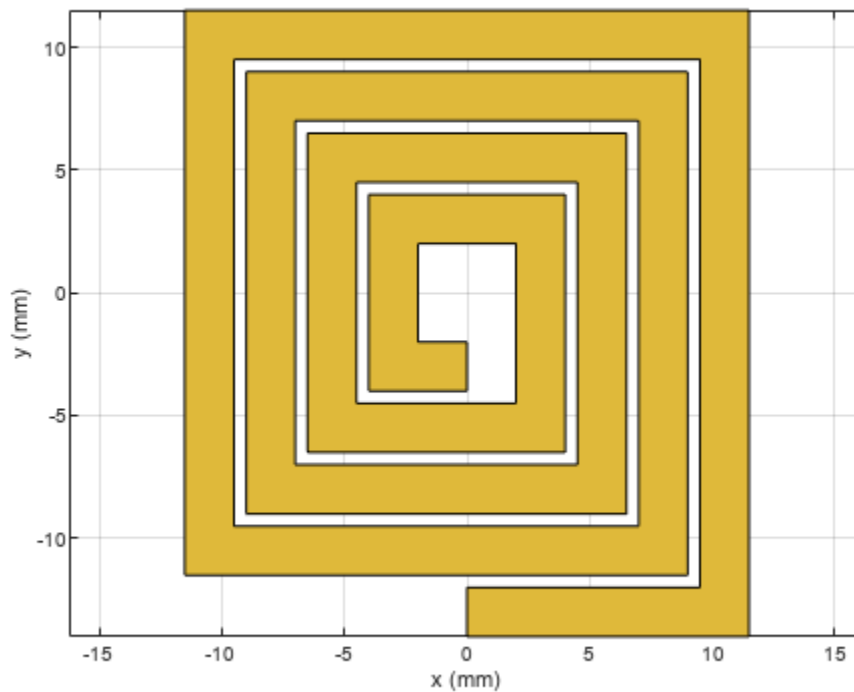
`c = rotateX(shape,angle)` rotates a shape by the specified angle about the x-axis.

Examples

Rotate Spiral Trace About X-Axis

Create and display a spiral trace.

```
trace = traceSpiral;  
show(trace)
```

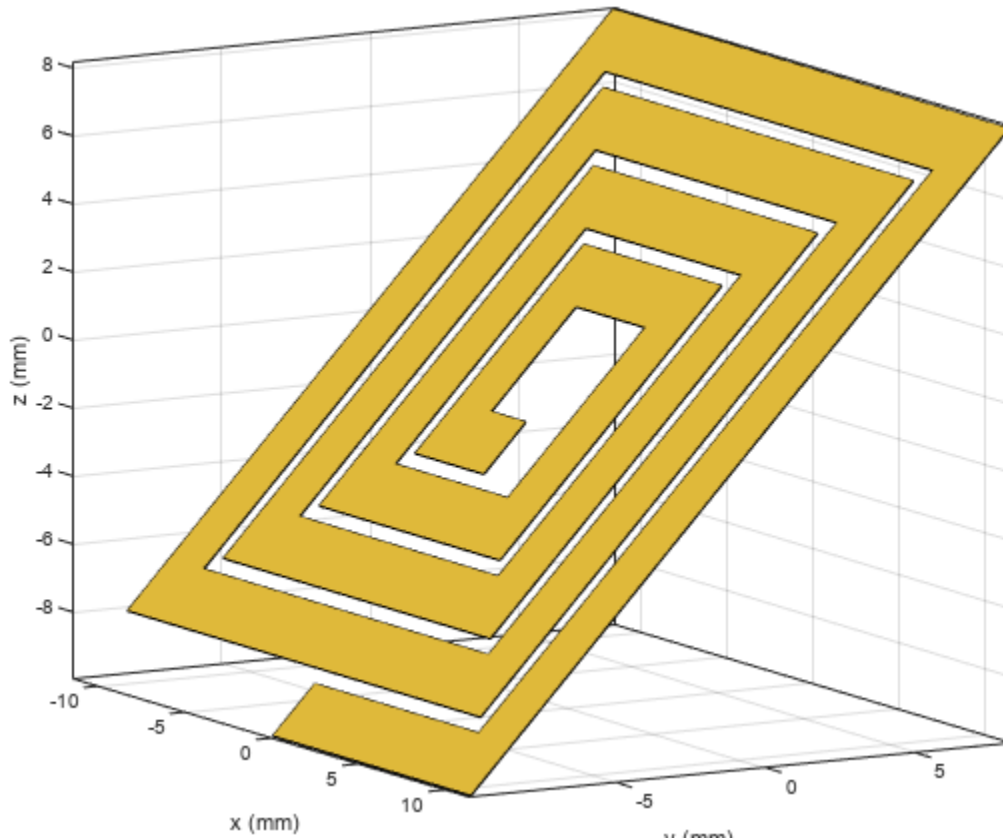


Rotate the spiral trace by 45 degrees about the x-axis.


```
traceRotX = rotateX(trace,45);
```

Display the rotated shape. Set the camera line of sight to display in 3-D space.

```
show(traceRotX)
view(60,10)
```



Input Arguments

shape — RF PCB shape

object

RF PCB shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape = bendCurved`; specifies the shape as a `bendCurved` object.

angle — Angle of rotation

scalar

Angle of rotation, specified as a scalar in degrees.

Example: 45 rotates the shape around the x-axis by 45 degrees.

Data Types: double

Version History

Introduced in R2021b

See Also

add | subtract | area | intersect | rotate | rotateY | rotateZ | translate | show | mesh |
plot | scale

rotateY

Rotate RF PCB shape about y-axis and angle

Syntax

```
c = rotateY(shape,angle)
```

Description

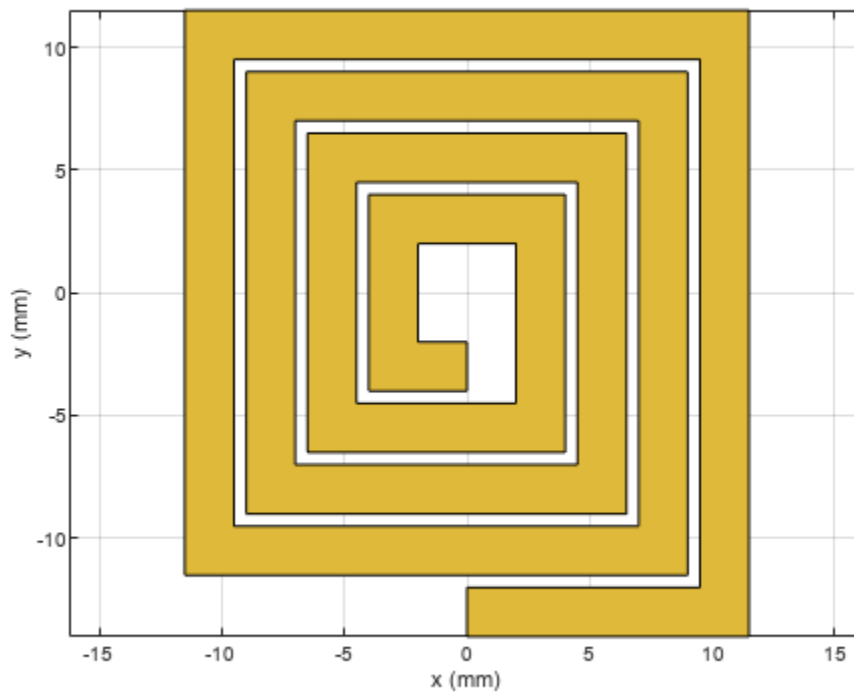
`c = rotateY(shape,angle)` rotates a shape by the specified angle about the y-axis.

Examples

Rotate Spiral Trace About Y-Axis

Create and display a spiral trace.

```
trace = traceSpiral;  
show(trace)
```

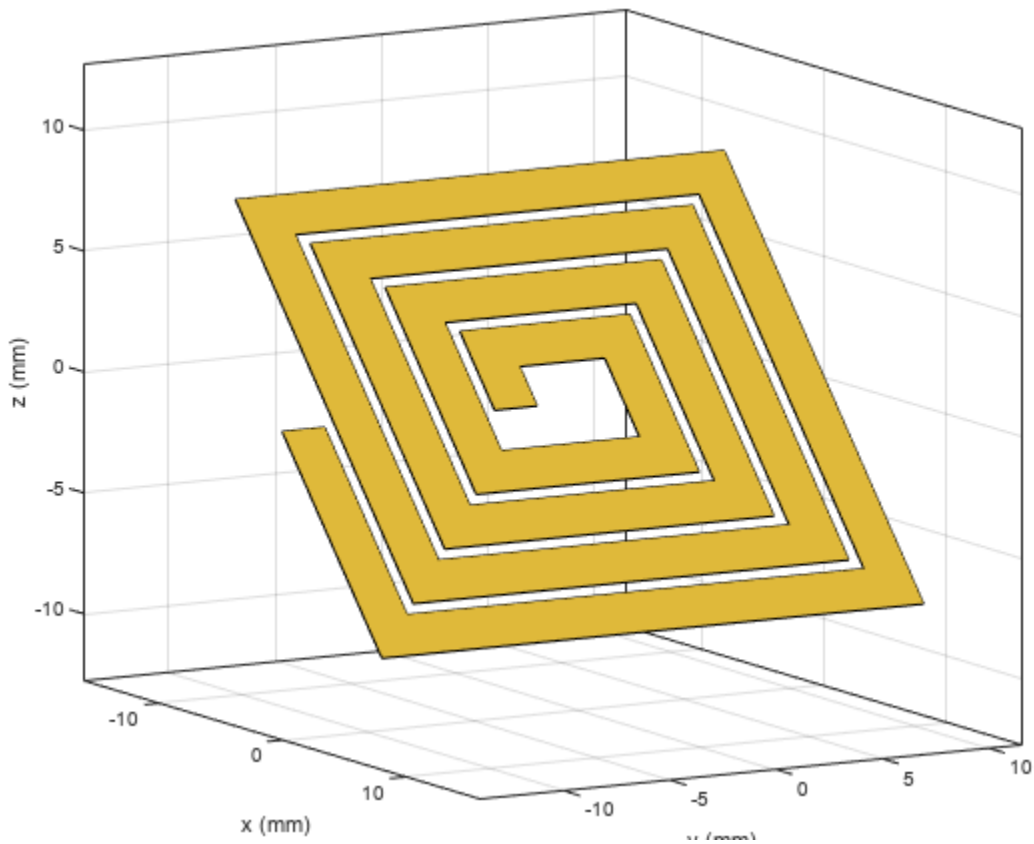


Rotate the spiral trace by 45 degrees about the y-axis.

```
traceRotY = rotateY(trace,45);
```

Display the rotated shape. Set the camera line of sight to display in 3-D space.

```
show(traceRotY)  
view(60,10)
```



Input Arguments

shape — RF PCB shape

object

RF PCB shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape = bendCurved`; specifies the shape as a `bendCurved` object.

angle — Angle of rotation

scalar

Angle of rotation, specified as a scalar in degrees.

Example: 45 rotates the shape around the y-axis by 45 degrees.

Data Types: `double`

Version History

Introduced in R2021b

See Also

add | subtract | area | intersect | rotate | rotateX | rotateZ | translate | show | mesh | plot | scale

rotateZ

Rotate RF PCB shape about z-axis

Syntax

```
c = rotateZ(shape,angle)
```

Description

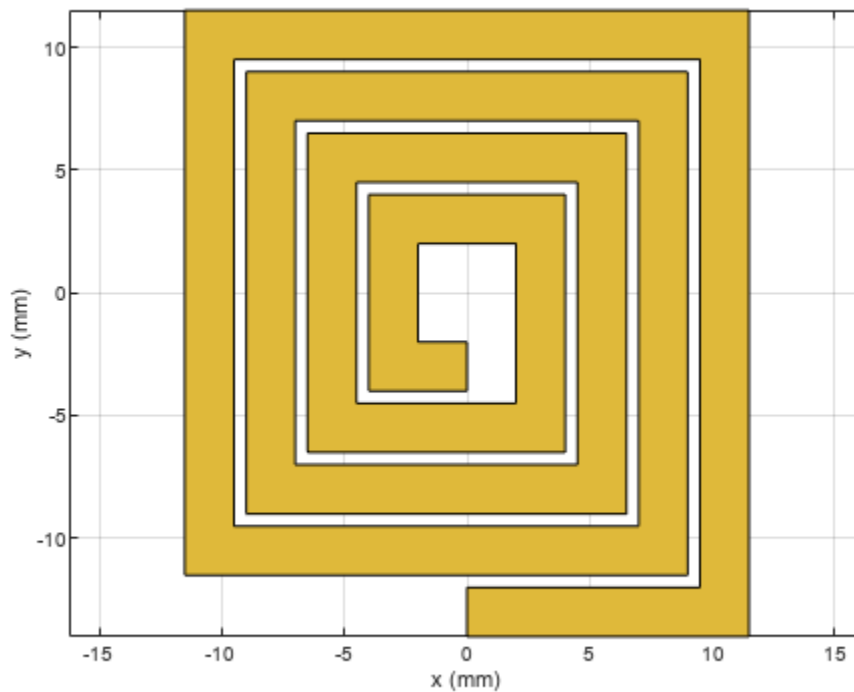
`c = rotateZ(shape,angle)` rotates a shape by the specified angle about the z-axis.

Examples

Rotate Spiral Trace About Z-Axis

Create and display a spiral trace.

```
trace = traceSpiral;  
show(trace)
```

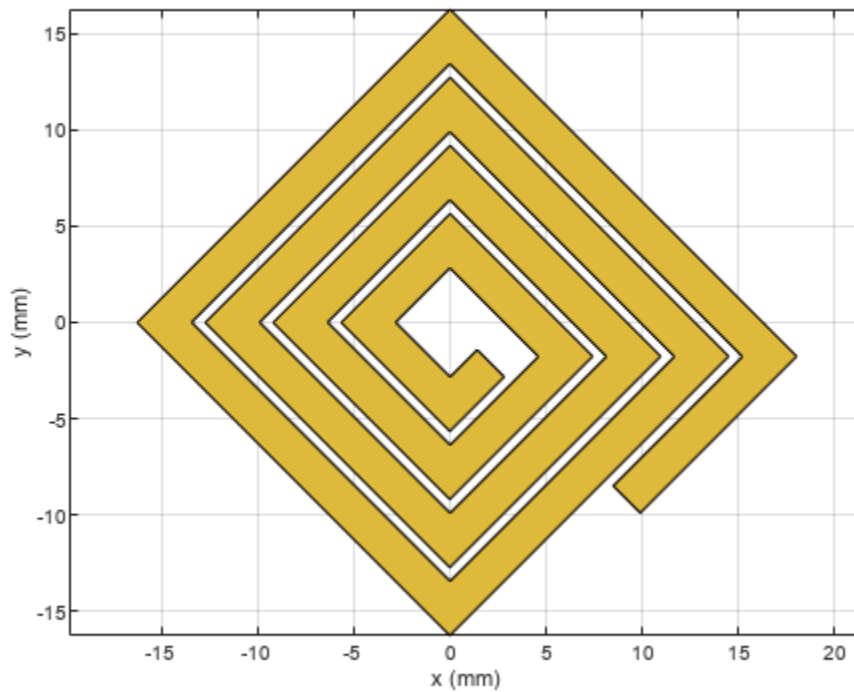


Rotate the spiral trace by 45 degrees about the z-axis.

```
traceRotZ = rotateZ(trace,45);
```

Display the rotated shape.

```
show(traceRotZ)
```



Input Arguments

shape — RF PCB shape

object

RF PCB shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape = bendCurved`; specifies the shape as a `bendCurved` object.

angle — Angle of rotation

scalar

Angle of rotation, specified as a scalar in degrees.

Example: `45` rotates the shape around the z-axis by 45 degrees.

Data Types: `double`

Version History

Introduced in R2021b

See Also

add | subtract | area | intersect | rotate | rotateX | rotateY | translate | show | mesh |
plot | scale

scale

Change size of RF PCB shape by fixed amount

Syntax

```
c = scale(shape, scaleFactor)
c = scale(shape, scaleFactor, RefPoint)
```

Description

`c = scale(shape, scaleFactor)` resizes the shape by a scaling factor.

`c = scale(shape, scaleFactor, RefPoint)` scales the shape by a constant factor with respect to the reference point. The reference point is ignored if the shape is symmetrical and scale is performed based on centroid. The reference point is considered if the shape is unsymmetrical and scale is performed based on specified reference point.

Examples

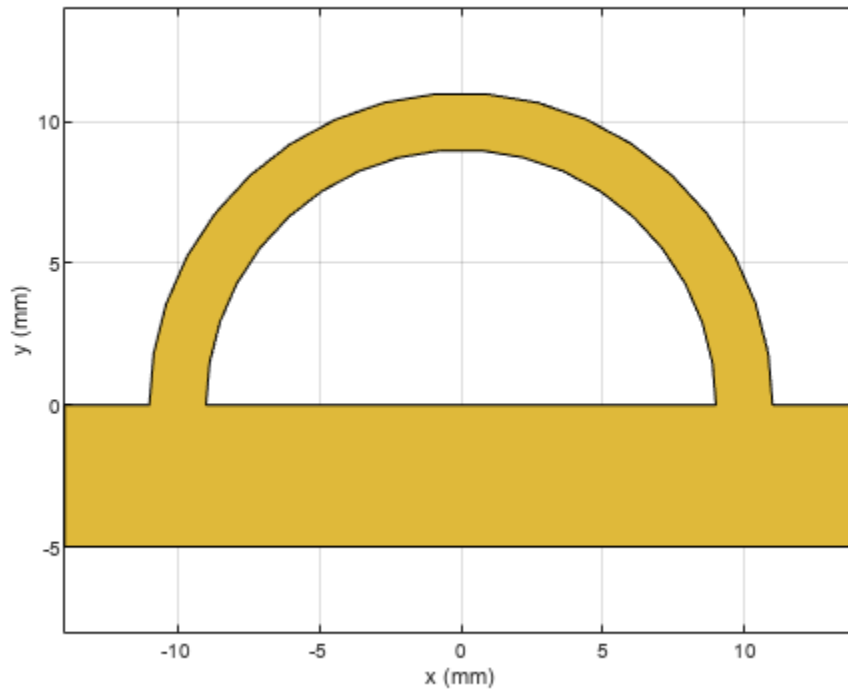
Resize Shape by Scale Factor

Create a shape consisting of a curve and a right angle U-bend.

```
shape1 = curve;
shape2 = ubendRightAngle(Length=[5 18 5]*1e-3,ReferencePoint=[0 -5]*1e-3);
shapeSum = shape1+shape2;
```

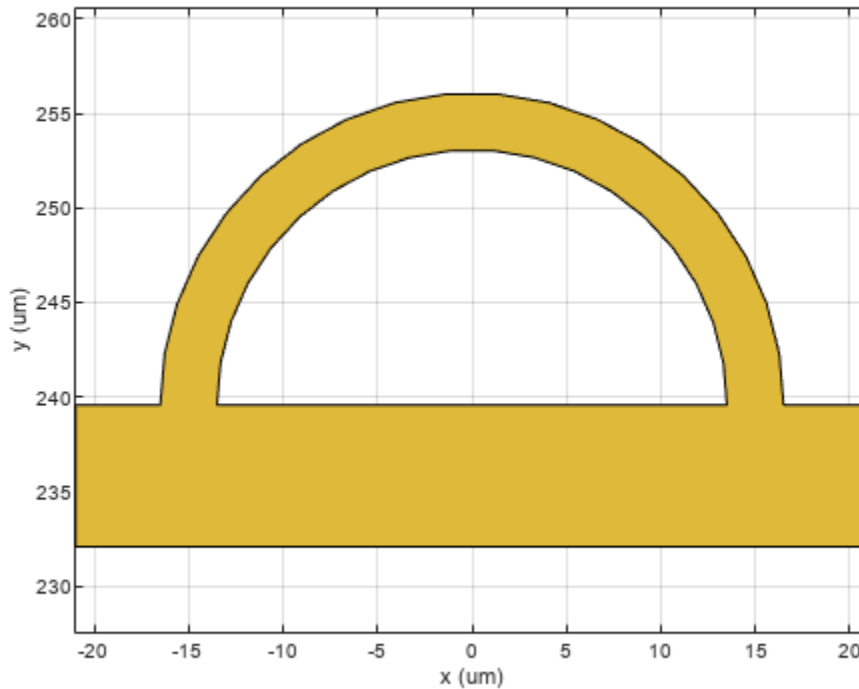
Display the shape.

```
show(shapeSum)
```



Specify a scale factor, then resize the shape. Display the result.

```
s = 1.5e-3;  
shapeTrans = scale(shapeSum,s);  
show(shapeTrans)
```



Input Arguments

shape — RF PCB shape

object

RF PCB shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape = bendCurved`; specifies the shape as a `bendCurved` object.

scaleFactor — Scaling factor

scalar

Scaling factor to change shape size, specified as a scalar.

Data Types: `double`

Version History

Introduced in R2021b

See Also

`add` | `subtract` | `area` | `intersect` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `show` | `mesh` | `plot`

subtract

Boolean subtraction operation on two RF PCB shapes

Syntax

```
c = subtract(shape1, shape2)
```

Description

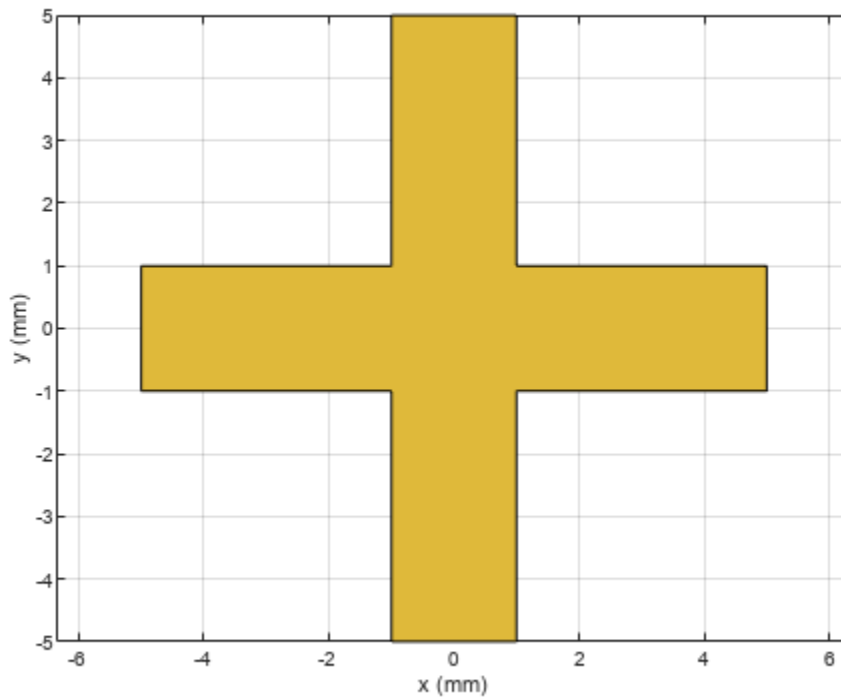
`c = subtract(shape1, shape2)` subtracts `shape1` and `shape2` using the subtract operation. You can also use the `-` symbol to subtract the two shapes.

Examples

Subtract Two RF PCB Shapes

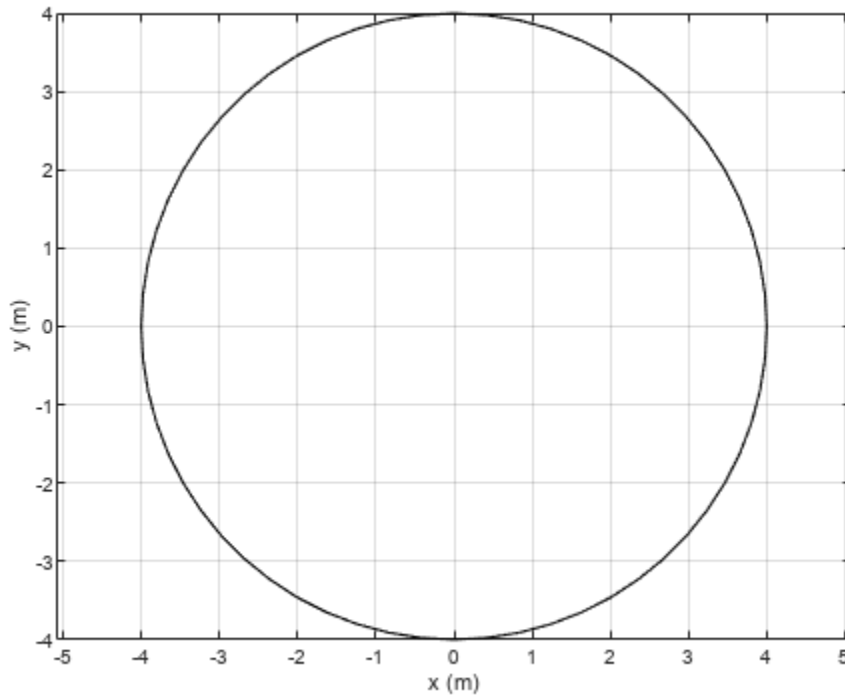
Create and display a cross trace shape.

```
trace = traceCross;  
show(trace)
```



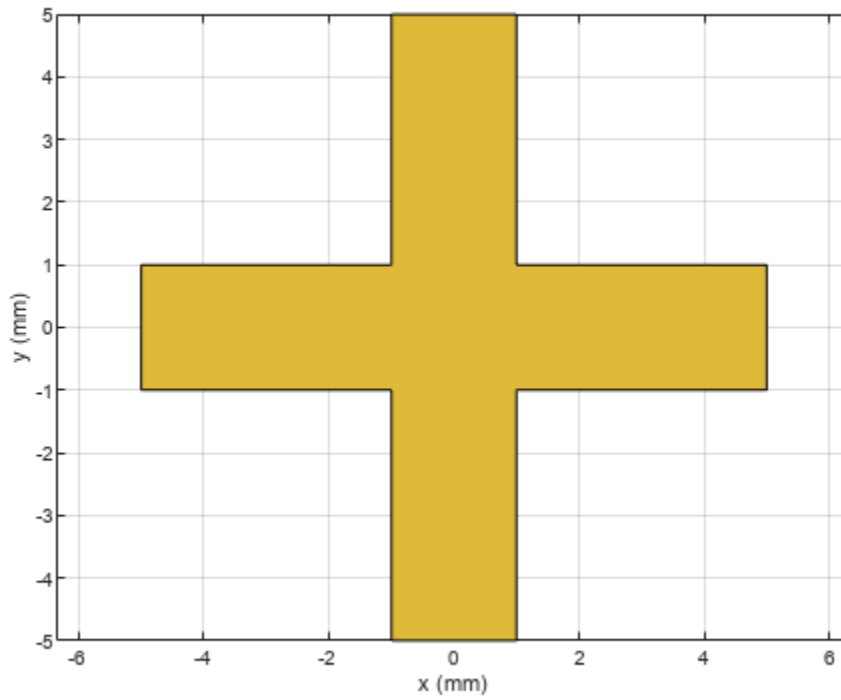
Create and display an annular ring shape with an inner radius of 4 m.

```
ring = ringAnnular(InnerRadius=4);  
show(ring)
```



Subtract the annular ring from the cross trace and display the result.

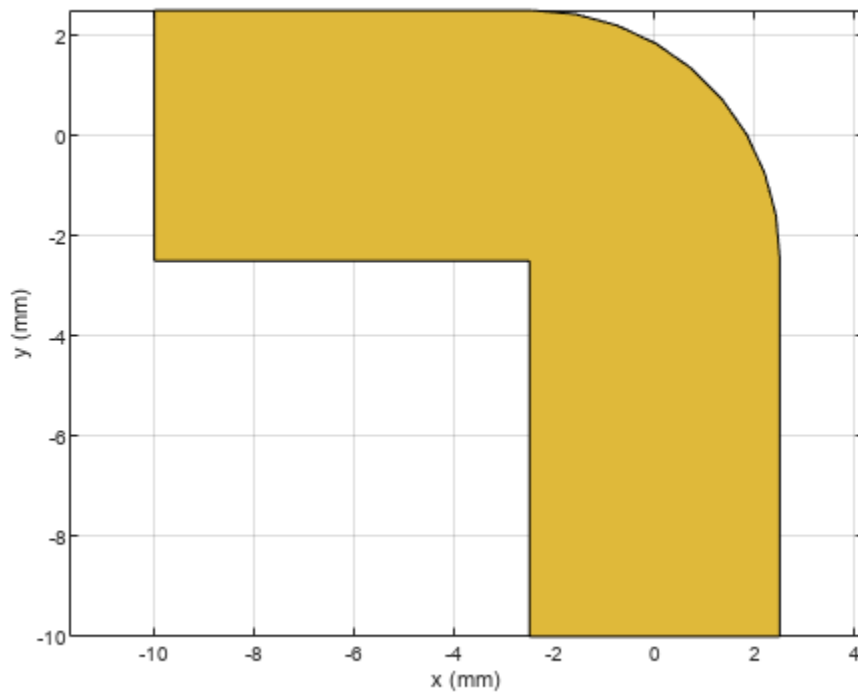
```
shapeDiff = subtract(trace,ring);  
show(shapeDiff)
```



Subtract Two RF PCB Shapes Using - Operator

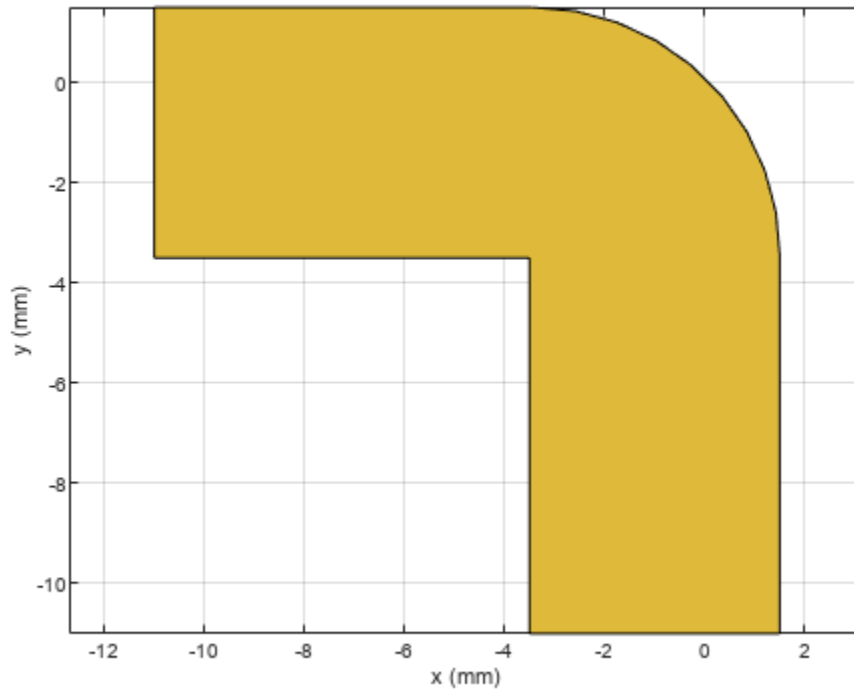
Create and display a curved bend shape.

```
bend1 = bendCurved;  
show(bend1)
```



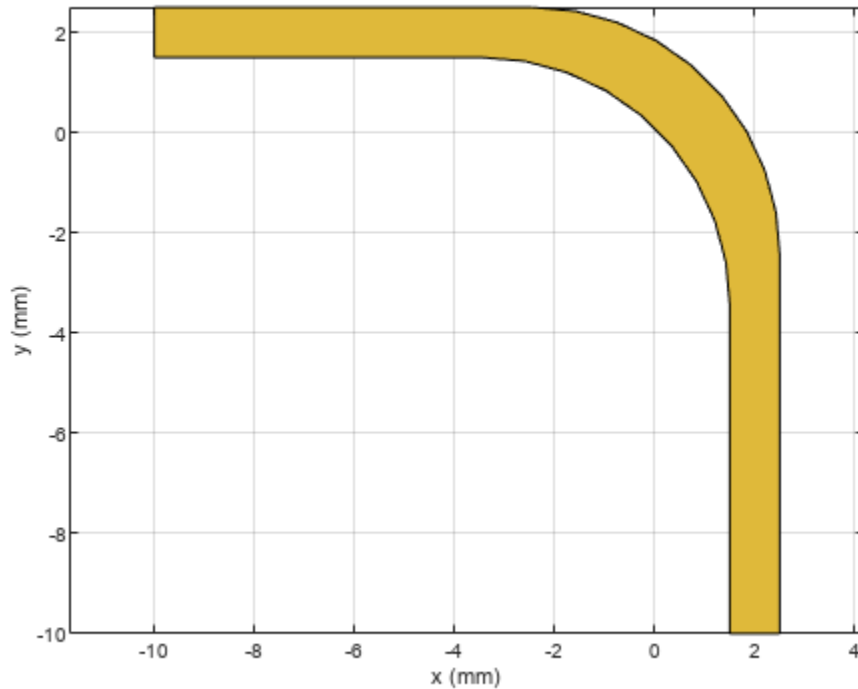
Create and display a curved bend shape with a spatial offset.

```
bend2 = bendCurved(ReferencePoint=[-1e-3 -1e-3]);  
show(bend2)
```



Subtract the offset bend from the default bend and display the result.

```
shapeDiff = bend1 - bend2;  
show(shapeDiff)
```

Input Arguments

shape1 — First shape
object

First shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape1 = bendCurved`; specifies the first shape as a `bendCurved` object.

shape2 — Second shape
object

Second shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape2 = ringAnnular`; specifies the second shape as a `ringAnnular` object.

Version History

Introduced in R2021b

See Also

add | area | intersect | rotate | rotateX | rotateY | rotateZ | translate | show | mesh |
plot | scale

translate

Move RF PCB shape to new location

Syntax

```
c = translate(shape,offset)
```

Description

`c = translate(shape,offset)` moves the shape to a new specified location using a translation vector.

Examples

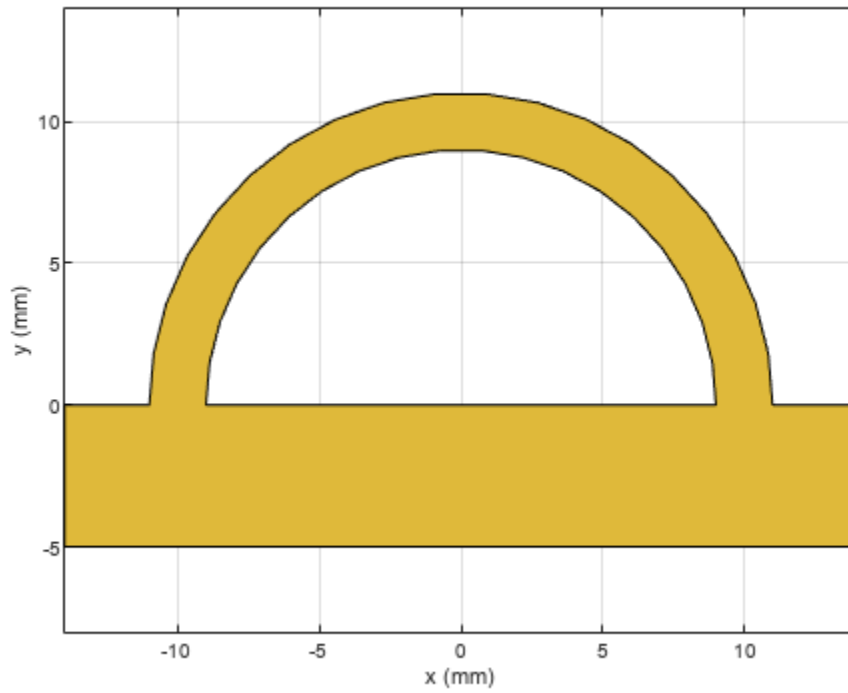
Translate Shape in XY Plane

Create a shape consisting of a curve and a right angle U-bend.

```
shape1 = curve;  
shape2 = ubendRightAngle(Length=[5 18 5]*1e-3,ReferencePoint=[0 -5]*1e-3);  
shapeSum = shape1+shape2;
```

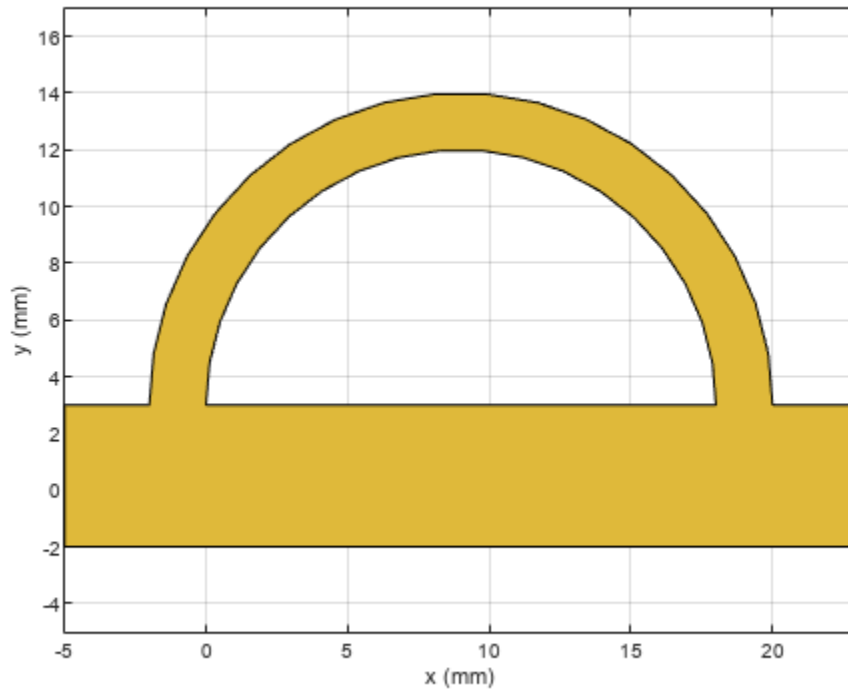
Display the shape.

```
show(shapeSum)
```



Specify a translation vector, then translate the shape in the X-Y plane. Display the result.

```
t = [9 3 0]*1e-3;  
shapeTrans = translate(shapeSum,t);  
show(shapeTrans)
```



Input Arguments

shape — RF PCB shape
object

RF PCB shape created using custom elements and shape objects of RF PCB Toolbox, specified as an object.

Example: `shape = bendCurved`; specifies the shape as a `bendCurved` object.

offset — Translation vector
vector

Translation vector, specified as a vector.

Data Types: `double`

Version History

Introduced in R2021b

See Also

add | subtract | area | intersect | rotate | rotateX | rotateY | rotateZ | show | mesh | plot
| scale

meshconfig

Change mesh mode of PCB component or shape structure

Syntax

```
meshconfig(rfpcbobject,mode)
meshconfig(shape,mode)
m = meshconfig( ____,mode)
```

Description

`meshconfig(rfpcbobject,mode)` changes the meshing mode of the PCB component according to the text input mode.

`meshconfig(shape,mode)` changes the meshing mode of the PCB shape according to the text input mode.

`m = meshconfig(____,mode)` returns the mesh structure after changing the meshing mode of the PCB component or shape according to the text input mode.

Examples

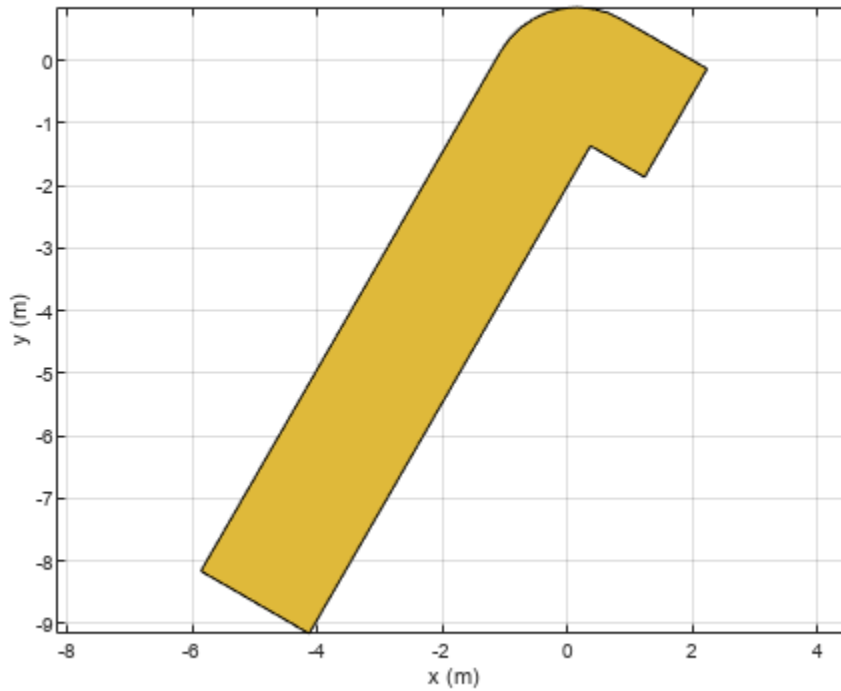
Mesh Rotated Curved Bend Shape

Create a curved bend shape of lengths of 10 m and 2 m, width of 2 m, and rotate it about the Z-axis by 60 degrees.

```
bend = bendCurved(Length=[10 2],Width=[2 2],CurveRadius=1)
```

```
bend =
  bendCurved with properties:
      Name: 'myCurvedbend'
  ReferencePoint: [0 0]
      Length: [10 2]
      Width: [2 2]
  CurveRadius: 1
```

```
bend = rotateZ(bend,60);
show(bend)
```

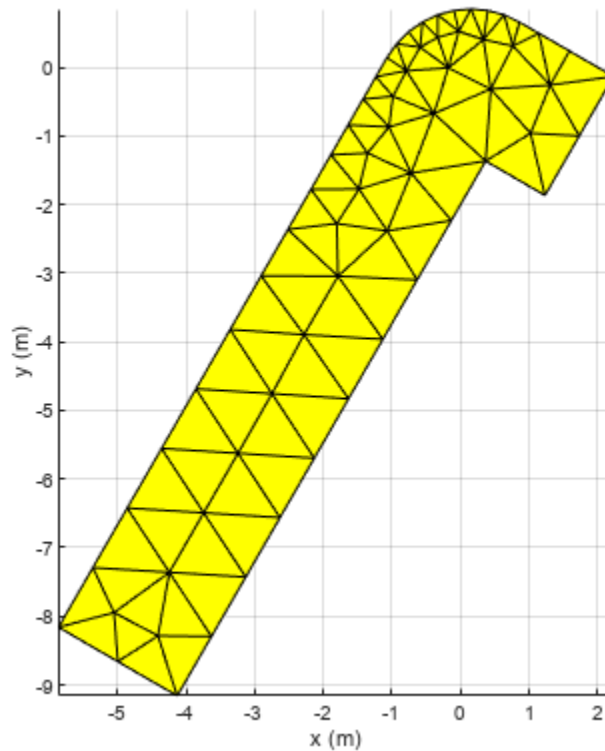


Mesh the curved bend shape at a maximum edge length of 1 m.

```
meshconfig(bend, "manual")
```

```
ans = struct with fields:  
    NumTriangles: 0  
    NumTetrahedra: 0  
    NumBasis: []  
    MaxEdgeLength: []  
    MinEdgeLength: []  
    GrowthRate: []  
    MeshMode: 'manual'
```

```
mesh(bend, MaxEdgeLength=1)
```

Input Arguments

rpcbobject — PCB component object

RF PCB object

PCB component object, specified as an RF PCB object. For a complete list of the PCB components, see “PCB Components Catalog”.

shape — Shape created using custom elements and shape objects

object handle

Shape created using custom elements and shape objects, specified as an object handle. For a complete list of the custom shapes, see “Custom Geometry and PCB Fabrication”.

Example: `c = bendCurved; mesh(c)`

mode — Meshing mode

'auto' (default) | 'manual'

Meshing mode, specified as 'auto' or 'manual'.

Data Types: char

Version History

Introduced in R2021b

See Also

mesh

info

Display information about PCB component structure

Syntax

```
info(rfpcbobject)
```

Description

`info(rfpcbobject)` displays information about the PCB component. as a structure:

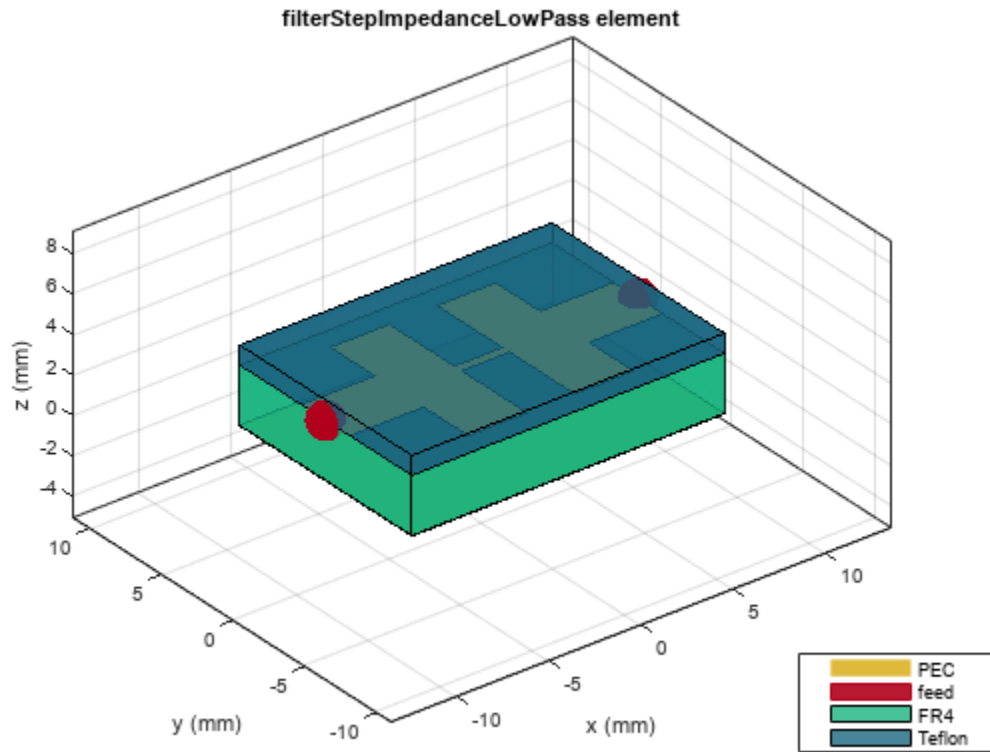
- `isSolved` - Logical specifying if an PCB component is solved.
- `isMeshed` - Logical specifying if an PCB component is meshed.
- `MeshingMode` - String specifying the meshing mode.
- `HasSubstrate` - Logical specifying if an PCB component uses a substrate.
- `HasLoad` - Logical specifying if an a PCB component has a load
- `PortFrequency` - Scalar or vector of frequencies used for port analysis.
- `FieldFrequency` - Scalar or vector of frequencies used for field analysis.
- `MemoryEstimate` - Approximate memory requirement for solving the antenna.

Examples

Create Stepped Impedance Lowpass Filter with Multilayer Dielectric Substrate

Create and view a stepped impedance lowpass filter with a multilayer dielectric substrate.

```
sub = dielectric("FR4", "Teflon");  
sub.Thickness = [0.003 0.001];  
steppedfilter = filterStepImpedanceLowPass;  
steppedfilter.Height = 0.003;  
steppedfilter.Substrate = sub;  
figure  
show(steppedfilter)
```



Plot the charge and current on the filter at 5 GHz.

```
figure  
charge(stepedfilter,5e9)
```

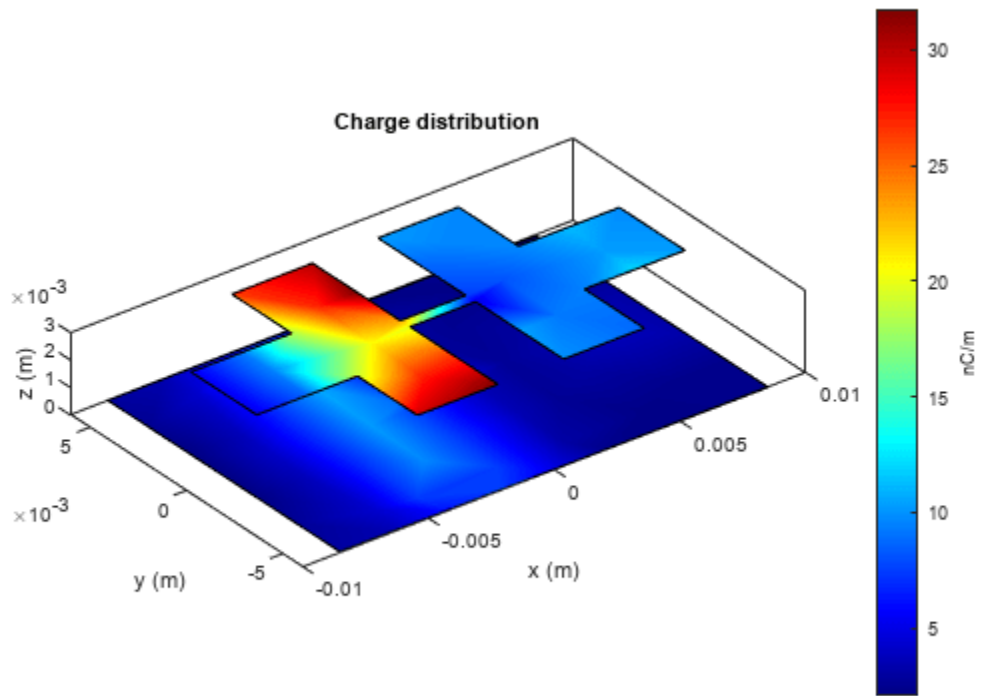
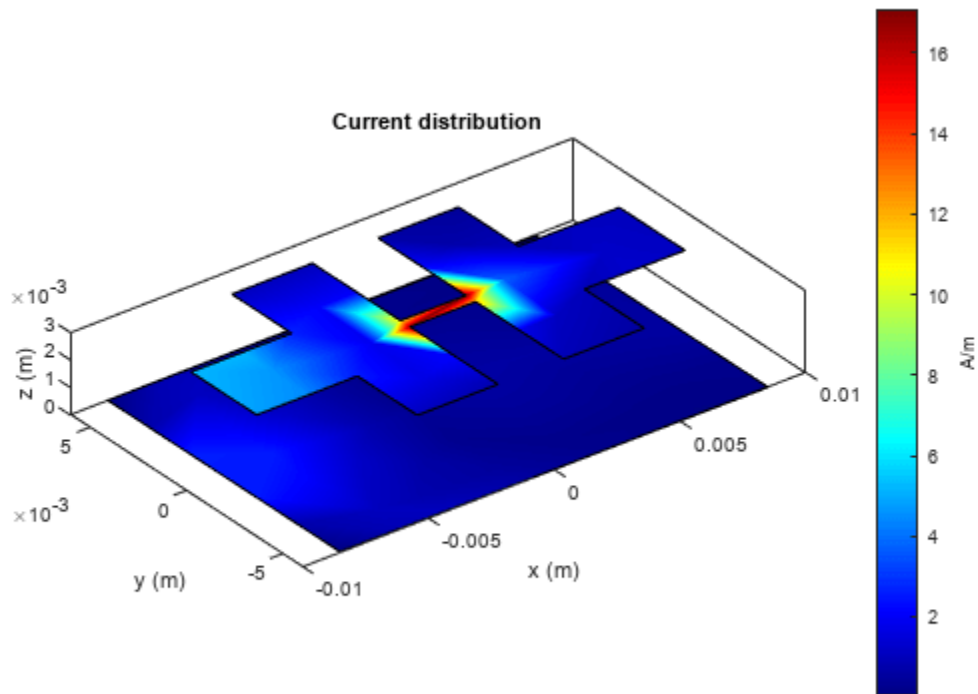


figure
current(steppedfilter,5e9)



```
info(stepperedfilter)
```

```
ans = struct with fields:
    IsSolved: "true"
    IsMeshed: "true"
    MeshingMode: "auto"
    HasSubstrate: "true"
    HasLoad: "false"
    PortFrequency: []
    MemoryEstimate: "790 MB"
```

Input Arguments

rfpcbobject — PCB component object

RF PCB object

PCB component object, specified as an RF PCB object. For a complete list of the PCB components, see “PCB Components Catalog”.

Version History

Introduced in R2021b

See Also

show

cylinder2strip

Cylinder equivalent width approximation

Syntax

```
w = cylinder2strip(r)
```

Description

`w = cylinder2strip(r)` calculates the equivalent width of a strip approximation for a cylindrical cross section.

Examples

Calculate Cylinder to Strip Approximation

Calculate the width of the strip approximation to a cylinder of radius 20 mm.

```
w = cylinder2strip(20e-3)
```

```
w = 0.0800
```

Input Arguments

r — Cylindrical cross-section radius

scalar | vector

Cylindrical cross-section radius, specified as a scalar or vector in meters.

Example: `20e-3`

Data Types: `double`

Output Arguments

w — Equivalent width of strip

scalar | vector

Equivalent width of strip, returned as a scalar or vector.

Data Types: `double`

Version History

Introduced in R2021b

See Also

voltagePort

Create voltage source with N-ports

Syntax

```
v = voltagePort(N)
v = voltagePort( ____,Name=Value)
```

Description

`v = voltagePort(N)` creates a voltage port source with *N* number of ports that you can use for excitation in an N-port PCB component.

`v = voltagePort(____,Name=Value)` creates a voltage port source using additional name-value arguments.

Examples

Create Voltage Source

Create a voltage source with three ports.

```
v = voltagePort(3)
v =
    voltagePort with properties:
        NumPorts: 3
        FeedVoltage: [1 0 0]
        FeedPhase: [0 0 0]
        PortImpedance: 50
```

Input Arguments

N — Number of ports

positive scalar

Number of ports, specified as a positive scalar.

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `FeedVoltage=1`

NumPorts — Number of ports

positive scalar

Number of ports, specified as a positive scalar.

FeedVoltage — Magnitude of voltage applied at each port

positive scalar | vector

Magnitude of the voltage applied at each port, specified as a positive scalar or vector.

FeedPhase — Phase shift applied to voltage at each port

positive scalar | vector

Phase shift applied to the voltage at each port in degrees, specified as a positive scalar or vector.

PortImpedance — Impedance to terminate each port

positive scalar | vector

Impedance to terminate each port in ohms, specified as a positive scalar or vector.

Version History

Introduced in R2021b

See Also

current | feedCurrent | charge

design

Design microstrip transmission line around specified frequency

Syntax

```
mline = design(mlineobj,frequency)
mline = design( ____,Name=Value)
```

Description

`mline = design(mlineobj,frequency)` designs a microstrip transmission line around the specified frequency.

`mline = design(____,Name=Value)` designs a microstrip transmission line with additional options specified using name-value arguments.

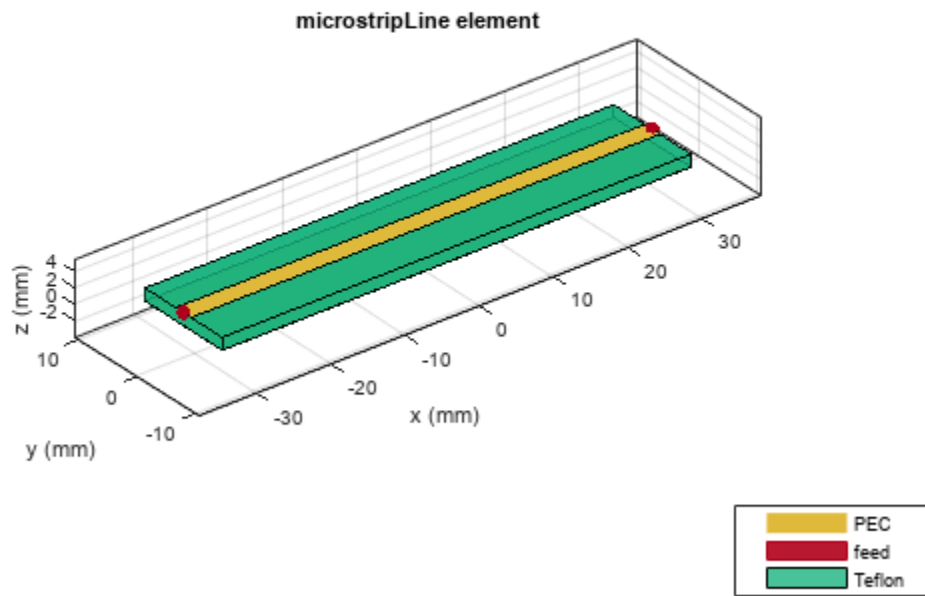
Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

Design Microstrip Transmission Line Around 1.8 GHz

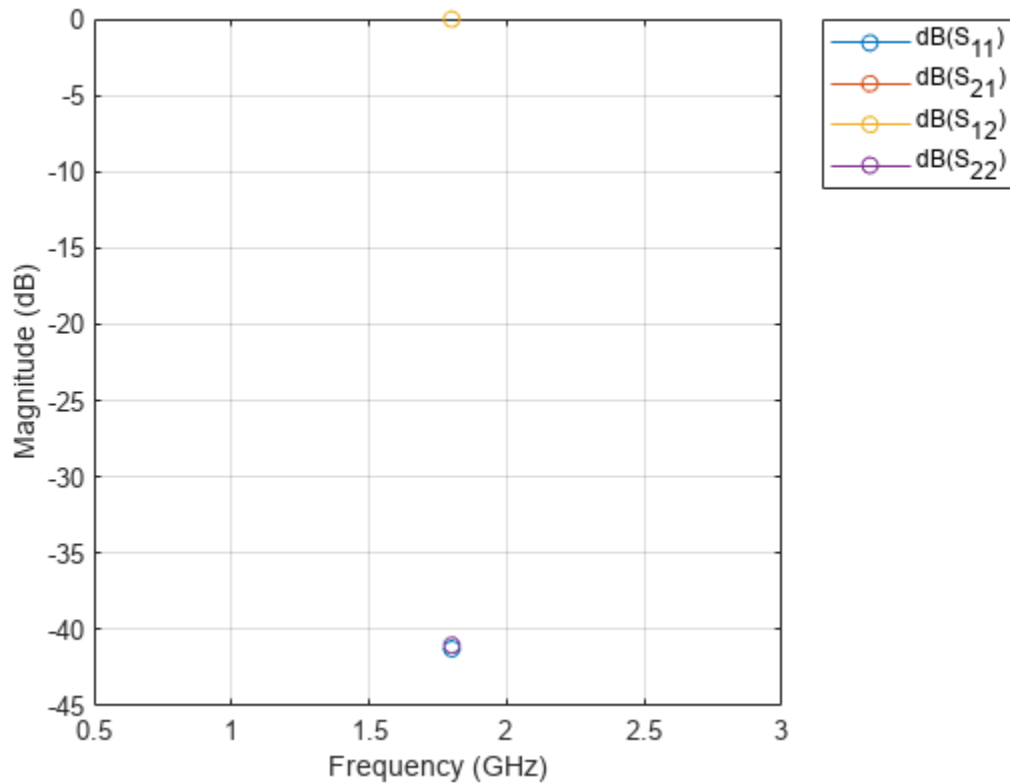
Design a microstrip line around 1.8 GHz and with a characteristic impedance of 75 ohms.

```
mline = design(microstripline,1.8e9,Z0=75);
figure;
show(mline);
```



Plot the S-parameters of this line.

```
spar = sparameters(mline,1.8e9);  
rfplot(spar)
```



Input Arguments

mlineobj – Microstrip transmission line

microstripLine object

Microstrip transmission line, specified as a microstripLine object.

Example: `mline = microstripLine; design(mline,2e9)` designs a microstrip transmission line around a frequency of 2 GHz.

frequency – Design frequency of transmission line

real positive scalar

Design frequency of the transmission line, specified as a real positive scalar in hertz.

Example: `55e6`

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=70`

Z0 — Characteristic impedance of microstrip transmission line

50 (default) | positive scalar

Characteristic impedance of the microstrip transmission line, specified as a positive scalar in ohms.

Data Types: double

LineLength — Length of line

0.5 (default) | positive scalar

Length of the line, specified as a positive scalar in Lambda.

Data Types: double

Output Arguments**mLine — Microstrip transmission line operating around specified frequency**

microstripLine object

Microstrip transmission line operating around the specified frequency, returned as a microstripLine object.

Version History**Introduced in R2021b****See Also**

sparameters

design

Design coplanar waveguide transmission line around particular frequency

Syntax

```
waveguide = design(cpw,frequency)
waveguide = design( ____,Name=Value)
```

Description

`waveguide = design(cpw,frequency)` designs an coplanar waveguide line around the specified frequency.

`waveguide = design(____,Name=Value)` designs a coplanar waveguide with additional options specified using name-value arguments.

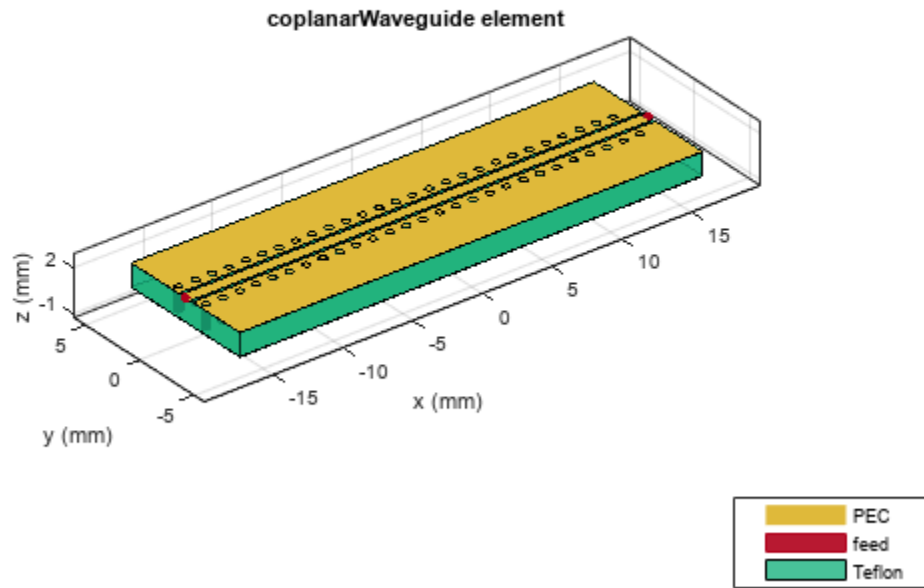
Note PCB components designed using the `design` function operates around the specified frequency with a 10-15% tolerance.

Examples

Design Coplanar Waveguide Around 1.8 GHz

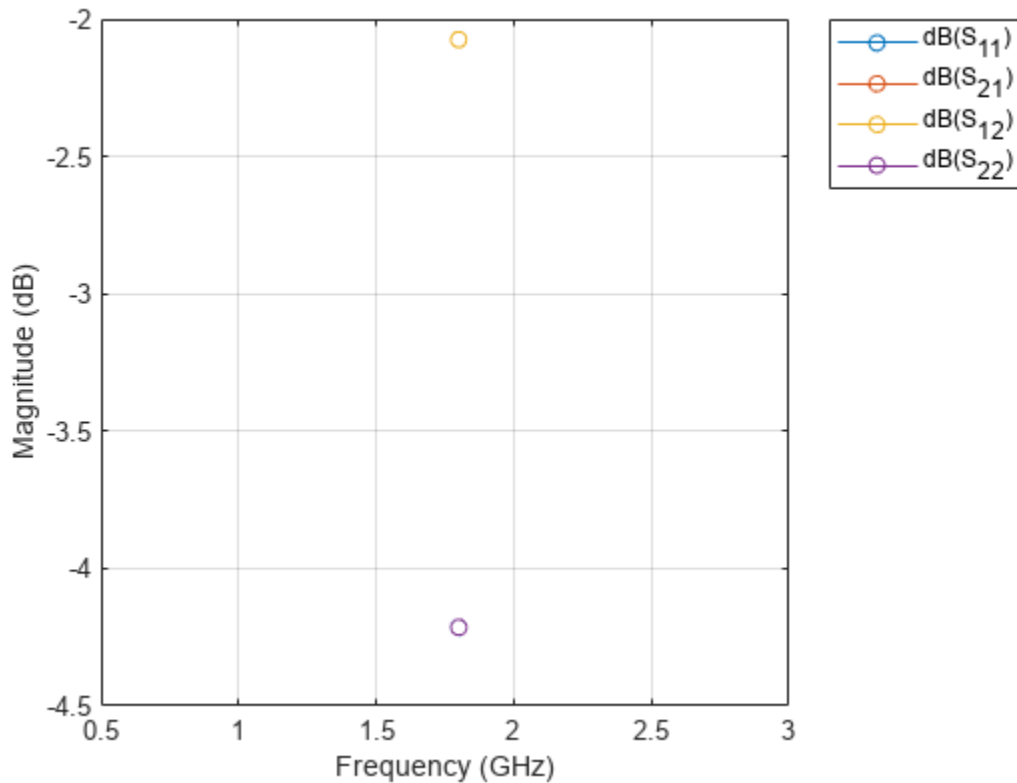
Design a lambda/4 coplanar waveguide at 1.8 GHz with a characteristic impedance of 75 ohms.

```
cpw = design(coplanarWaveguide, 1.8e9,Z0=75,LineLength=0.25);
figure;
show(cpw);
```



Plot the s-parameters of the waveguide at 1.8 GHz.

```
spar = sparameters(cpw,1.8e9);  
rfplot(spar)
```

Input Arguments

cpw — Coplanar waveguide

coplanarWaveguide object

Coplanar waveguide transmission line, specified as a `coplanarWaveguide` object.

Example: `waveguide = coplanarWaveguide; waveguide = design(waveguide,2e9)` designs a coplanar waveguide around a frequency of 2 GHz.

frequency — Design frequency of transmission line

real positive scalar

Design frequency of the transmission line, specified as a real positive scalar in hertz.

Example: `3e9`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=75`

Z0 — Characteristic impedance of line

50 (default) | positive scalar

Characteristic impedance of the line, specified as a positive scalar in ohms.

Data Types: double

LineLength — Length of coplanar waveguide

0.5 (default) | positive scalar

Length of the coplanar waveguide, specified as a positive scalar in terms of Lambda.

Data Types: double

Output Arguments**waveguide — Coplanar waveguide operating around specified reference frequency**

coplanarWaveguide object

Coplanar waveguide operating around the specified reference frequency, returned as a coplanarWaveguide object.

Version History**Introduced in R2021b****See Also**

sparameters

design

Design coupled microstrip transmission line around particular frequency

Syntax

```
cline = design(clineobj,frequency)
cline = design( ____,Name=Value)
```

Description

`cline = design(clineobj,frequency)` designs a coupled microstrip transmission line around the specified frequency.

`cline = design(____,Name=Value)` designs a coupled microstrip transmission line with additional options specified using name-value arguments.

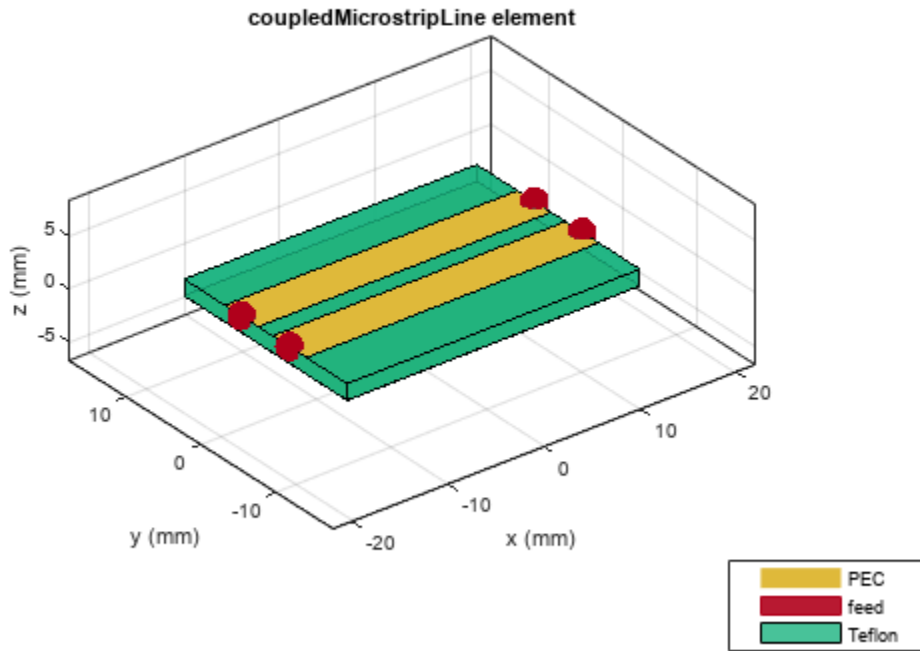
Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

Design Coupled Microstrip Transmission Line Around 1.8 GHz

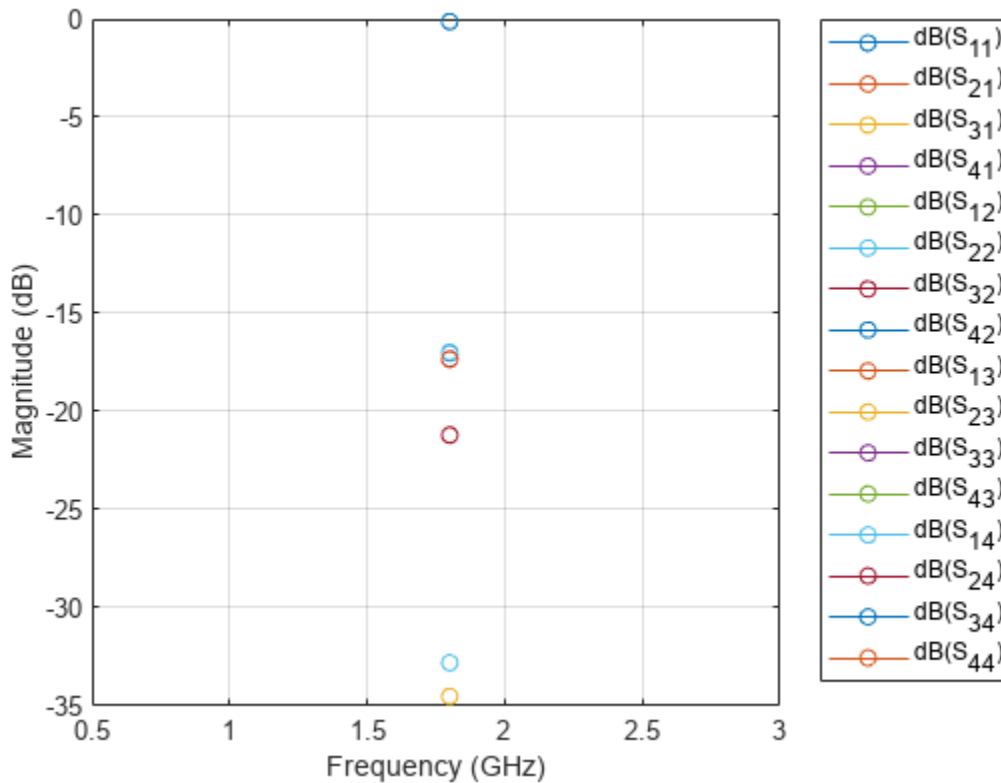
Design a coupled microstrip line at 1.8 GHz with an even mode impedance of 60 ohms.

```
cline = design(coupledMicrostripLine,1.8e9,Z0e=60);
show(cline);
```



Plot the S-parameters of the transmission line.

```
spar = sparameters(cline,1.8e9);  
rfplot(spar)
```



Input Arguments

clineobj – Coupled microstrip transmission line

coupledMicrostripLine object

Coupled microstrip transmission line, specified as a coupledMicrostripLine object.

Example: `cline = coupledMicrostripLine; design(cline,2e9)` designs a coupled microstrip transmission line around a frequency of 2 GHz.

frequency – Design frequency of transmission line

real positive scalar

Design frequency of the transmission line, specified as a real positive scalar in hertz.

Example: `2e9`

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0e=45`

Z0e – Even mode impedance

52 (default) | positive scalar

Even mode impedance of the transmission line, specified as a positive scalar in ohms.

Data Types: double

Z0o – Odd mode impedance

48 (default) | positive scalar

Odd mode impedance of the transmission line, specified as a positive scalar in ohms.

Data Types: double

Output Arguments**cLine – Coupled microstrip transmission line operating around specified frequency**

coupledMicrostripLine object

Coupled microstrip transmission line operating around the specified frequency, returned as a coupledMicrostripLine object.

Version History**Introduced in R2021b****See Also**

sparameters

design

Design branchline coupler around particular frequency

Syntax

```
coupler = design(couplerobj,frequency)
coupler = design( ___,Name,Value)
```

Description

`coupler = design(couplerobj,frequency)` designs a branchline coupler around the specified frequency.

`coupler = design(___,Name,Value)` designs a branchline coupler line with additional options specified using name-value arguments.

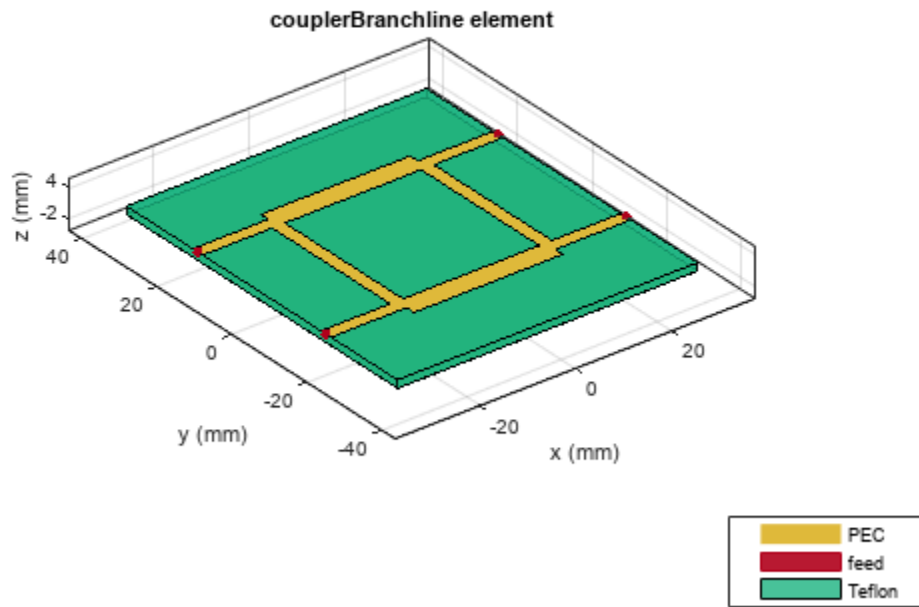
Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

Design Branchline Coupler Around 1.8 GHz

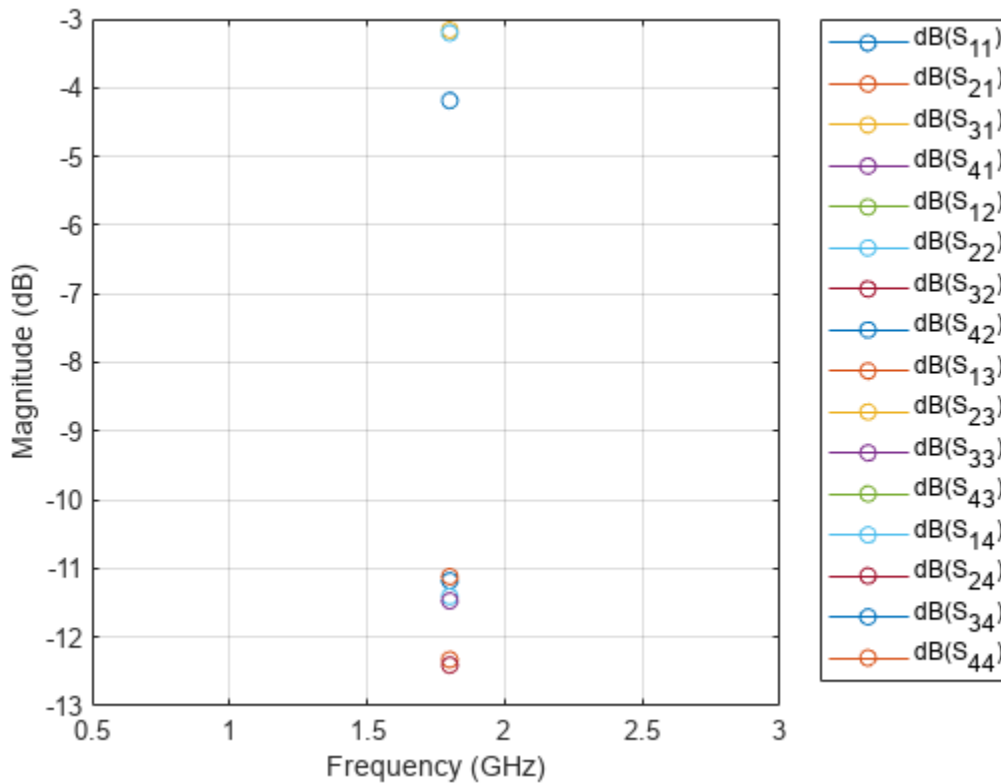
Design a branchline coupler at 1.8 GHz and with a Z_0 of 75 ohms.

```
coupler = design(couplerBranchline,1.8e9,Z0=75);
show(coupler);
```



Plot the s-parameters of the coupler at 1.8 GHz.

```
spar = sparameters(coupler,1.8e9);  
rfplot(spar)
```

Input Arguments

couplerobj – Branchline coupler

couplerBranchline object

Branchline coupler, specified as a couplerBranchline object.

Example: `coupler = couplerBranchline; design(coupler,2e9)` designs a branchline coupler around a frequency of 2 GHz.

frequency – Design frequency of coupler

real positive scalar

Design frequency of the coupler, specified as a real positive scalar in hertz.

Example: `3e9`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=70`

Z0 — Characteristic impedance of coupler

50 (default) | positive scalar

Characteristic impedance of the coupler, specified as a positive scalar in ohms.

Data Types: double

Output Arguments**coupler — Branchline coupler operating around specified frequency**

coupleBranchline object

Branchline coupler operating around the specified frequency, returned as a coupleBranchline object.

Version History**Introduced in R2021b****See Also**

sparameters

design

Design rat-race coupler around specified frequency

Syntax

```
coupler = design(couplerobj,frequency)
coupler = design( ____,Name=Value)
```

Description

`coupler = design(couplerobj,frequency)` designs a rat-race coupler around the specified frequency.

`coupler = design(____,Name=Value)` designs a rat-race coupler line with additional options specified using name-value arguments.

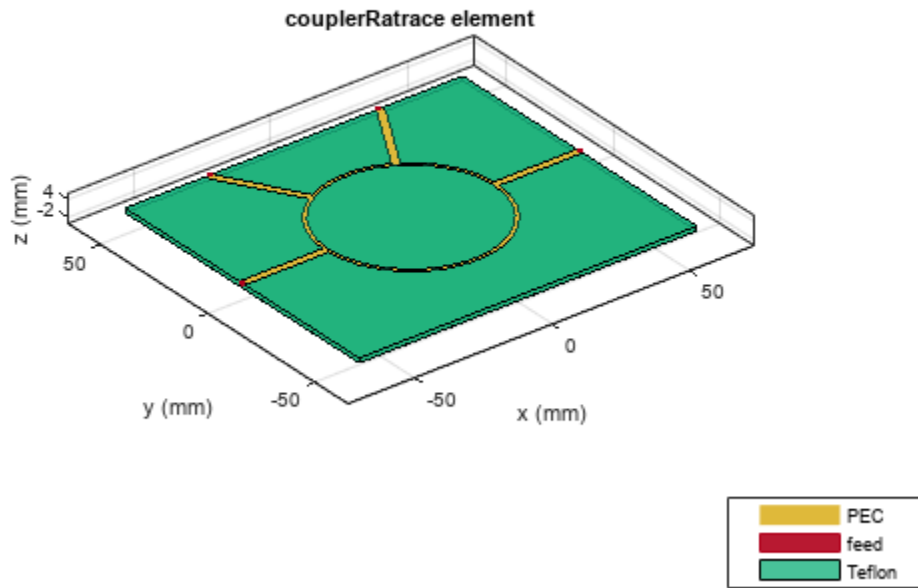
Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

Design Rat-race Coupler Around 1.8 GHz

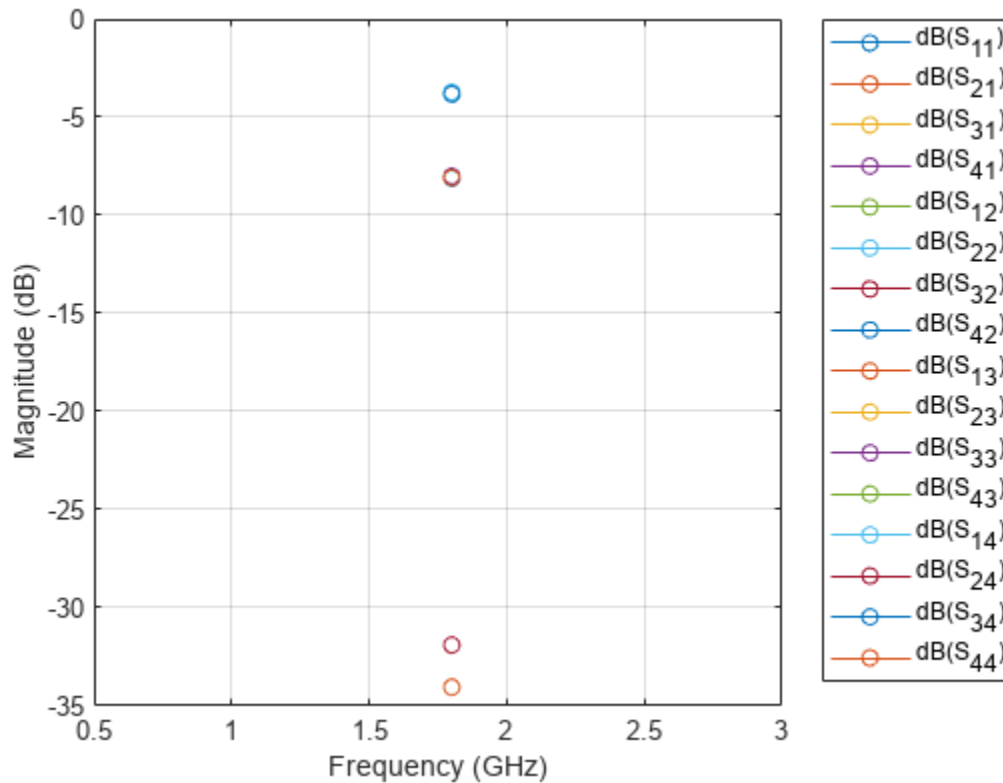
Design a rat-race coupler at 1.8 GHz and with a characteristic impedance of 75 ohms.

```
coupler = design(couplerRatrace,1.8e9,Z0=75);
show(coupler);
```



Plot the S-parameters of the coupler at 1.8 GHz.

```
spar = sparameters(coupler,1.8e9);  
rfplot(spar)
```



Input Arguments

couplerobj — Rat-race coupler

couplerRatrace object

Rat-race coupler, specified as a couplerRatrace object.

Example: `coupler = couplerRatrace; design(coupler,2e9)` designs a rat-race coupler around a frequency of 2 GHz.

frequency — Design frequency of rat-race coupler

real positive scalar

Design frequency of the rat-race coupler, specified as a real positive scalar in hertz.

Example: `2.5e9`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=70`

Z0 — Characteristic impedance of coupler

50 (default) | positive scalar

Characteristic impedance of the coupler in ohms, specified as a positive scalar.

Data Types: double

Output Arguments**coupler — Rat-race coupler operating around specified reference frequency**

couplerRatrace object

Rat-race coupler operating around the specified reference frequency, returned as a couplerRatrace object.

Version History**Introduced in R2021b****See Also**

sparameters

sparameters

Calculate S-parameters for RF PCB objects

Syntax

```
sobj = sparameters(rfpcbobj, freq)
sobj = sparameters( ____, Z0)
sobj = sparameters( ____, Name=Value)

sobj = sparameters(data, freq)
sobj = sparameters(data, freq, Z0)

sobj = sparameters(filename)
```

Description

`sobj = sparameters(rfpcbobj, freq)` calculates the S-parameters for the RF PCB object `rfpcbobj` over the specified frequency values.

`sobj = sparameters(____, Z0)` calculates the S-parameters for the reference impedance `Z0`.

`sobj = sparameters(____, Name=Value)` calculates S-parameters using one or more name-value arguments in addition to any of the input argument combinations in previous syntaxes

`sobj = sparameters(data, freq)` creates an S-parameter object from the S-parameter data provided in `data` over the specified frequencies values.

`sobj = sparameters(data, freq, Z0)` creates an S-parameter object for the reference impedance `Z0`.

`sobj = sparameters(filename)` creates an S-parameter object from the data provided in the Touchstone file specified in `filename`.

Examples

Calculate S-parameters for Wilkinson Power Splitter

Create a Wilkinson power splitter object.

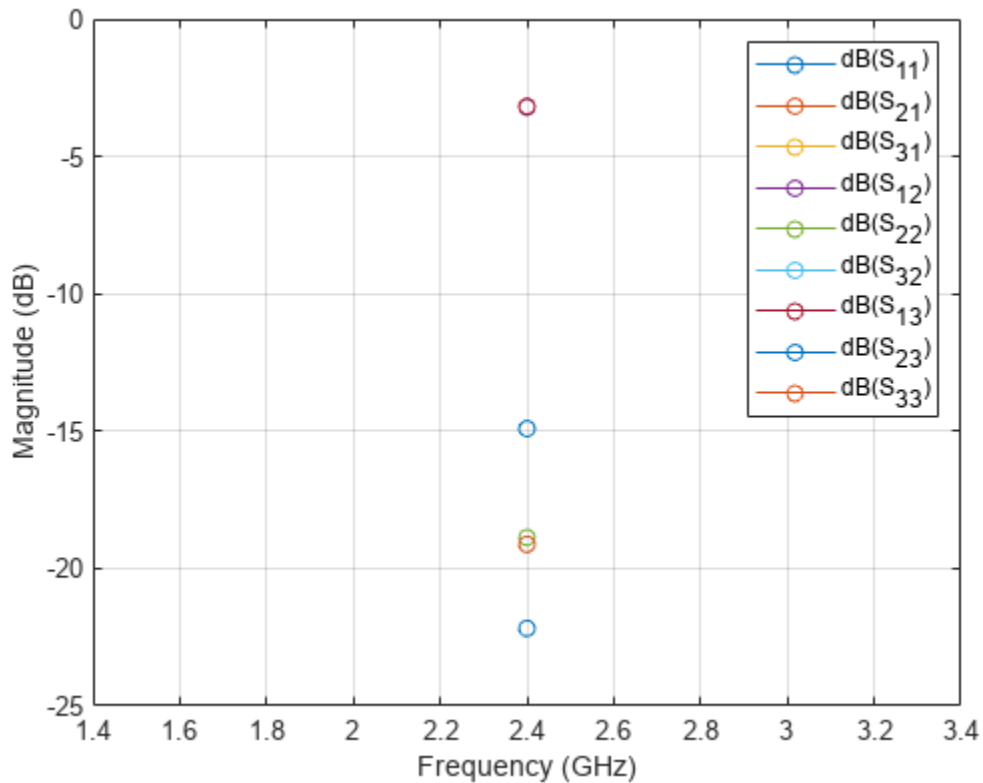
```
rfpcbobj = wilkinsonSplitter;
```

Calculate the S-parameters for the Wilkinson power splitter at 2.4 GHz with the reference impedance of 50 ohms.

```
Sobj = sparameters(wilkinsonSplitter, 2.4e9, 50);
```

Plot the S-parameters using the `rfplot` function.

```
rfplot(Sobj)
```



Calculate S-Parameters of Right Angle Bend

Design a microstrip transmission line at 3 GHz with 75 ohms impedance.

```
m = microstripline(Length=0.0379,Width=0.0027,Height=0.0016,GroundPlaneWidth=0.0133);
```

Create a right angle bend with length equal to half the length of the transmission line and width equal to the width of the transmission line.

```
layer2d = bendRightAngle( Length=[m.Length/2 m.Length/2], ...
Width=[m.Width m.Width]);
```

Convert the right angle bend to a 3-D component.

```
robject = pcbComponent(layer2d);
```

Add thickness and substrate layers to the board.

```
robject.BoardThickness = m.Substrate.Thickness;
robject.Layers{2} = m.Substrate;
```

Define frequency points to calculate the S-parameters.

```
freq = (1:2:40)*100e6;
```

Calculate the S-parameters of the right angle bend using the behavioral model.

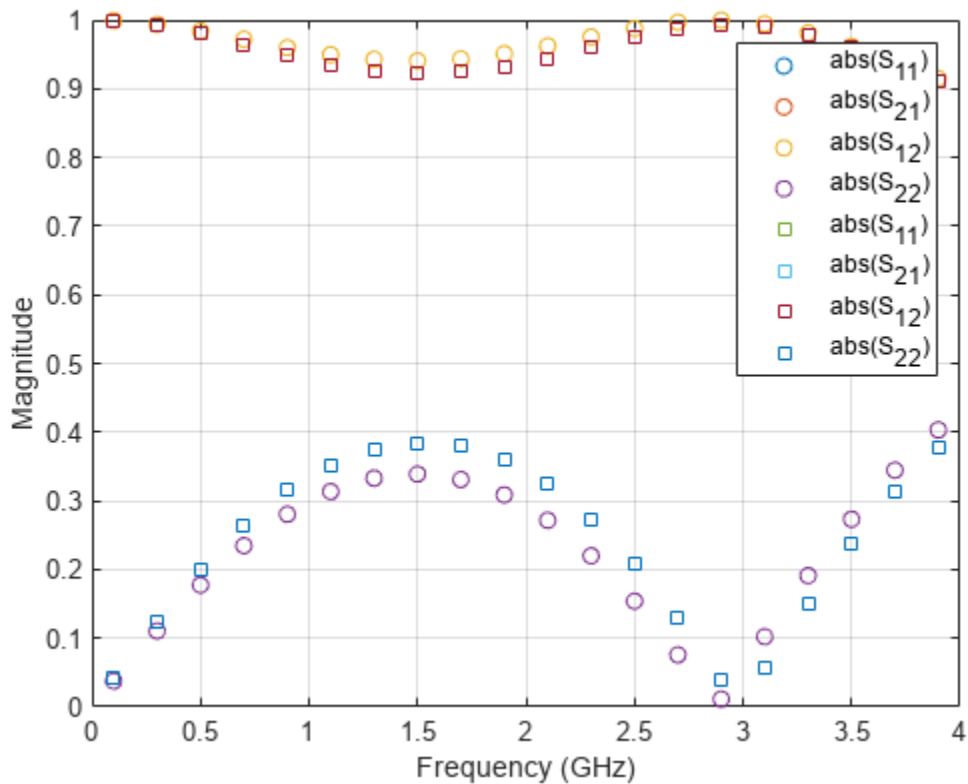

```
Sckt1 = sparameters(roboj, freq, Behavioral=true);
```

Calculate the S-parameters of the right angle bend using the electromagnetic solver.

```
Sem1 = sparameters(roboj, freq);
```

Plot the S-parameter data using the rfplot function.

```
rfplot(Sckt1, 'abs', 'o')
hold on
rfplot(Sem1, 'abs', 's')
```



Input Arguments

rfpcbobj — Input object

RF PCB object

Input object, specified as a RF PCB object. You can specify either a PCB component, a microstrip bend, or a trace. For complete list of PCB components, microstrip bends, and traces, see “PCB Components Catalog” and “Custom Geometry and PCB Fabrication”.

data — S-parameter data

array of complex numbers

S-parameter data, specified as an array of complex numbers of the size N -by- N -by- K , where K represents number of frequency points.

freq — S-parameter frequency

scalar | vector

S-parameter frequencies, specified as a scalar or vector of positive real numbers in the ascending order.

Z0 — Reference impedance

50 (default) | positive real scalar

Reference impedance in ohms, specified as a positive real scalar.

filename — Name of Touchstone file

character vector | string scalar

Name of the Touchstone file containing network parameter data, specified as a character vector or string scalar. If the file is in the current folder or in a folder on the MATLAB® path, specify the file name. If the file is not in the current folder or in a folder on the MATLAB path, then specify the full or relative path name.

Example: `sobj = sparameters('defaultbandpass.s2p');`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name in quotes.

Example: `Sobj = sparameters(robj,freq,Behavioral=true)`

Behavioral — Behavioral model of RF PCB component and bend

false or 0 (default) | true or 1

Behavioral model of an RF PCB component and bend, specified as true (1) or false (0). You can compute the behavioral model for these `rfpcb` objects:

Bends	<ul style="list-style-type: none"> • bendRightAngle • bendCurved • bendMitered
Traces	<ul style="list-style-type: none"> • traceTee • traceCross
Transmission line objects	<ul style="list-style-type: none"> • microstripLine • coplanarWaveguide
Inductor	spiralInductor
Capacitor	interdigitalCapacitor

Note

- Before using the `sparameters` function to calculate S-parameters for bends and traces, convert bends and traces to PCB components using the `pcbComponent` function.

- The `sparameters` function does not support using the behavioral model argument for:
 - Objects with unequal widths like `bendRightAngle`, `bendCurved`, and `bendMitered`
 - Asymmetric tee and cross-junction traces
-

Example: `Sobj = sparameters(microstripline,freq,Behavioral = true)`

Output Arguments

sobj — S-parameters

S-parameter object

S-parameters, returned as an object with the following properties:

- `NumPorts` — Number of ports, N , returned as an integer. The function calculates this value automatically when you create the object.
- `Frequencies` — S-parameter frequency, returned as a scalar or row vector of length, K , in the ascending order. The function sets this property from the `filename` or `freq` input arguments.
- `Parameters` — S-parameters, returned as an N -by- N -by- K array of complex numbers. The function sets this property from the `filename` or `data` input arguments.
- `Impedance` — Reference impedance in ohms, returned as a positive real scalar. The function sets this property from the `filename` or `Z0` input arguments. If you do not provide reference impedance,, the function uses a default value of 50.

Version History

Introduced in R2021b

See Also

`current` | `getZ0`

layout

Plot all metal layers and board shape

Syntax

```
layout(rfpcbobject)
```

Description

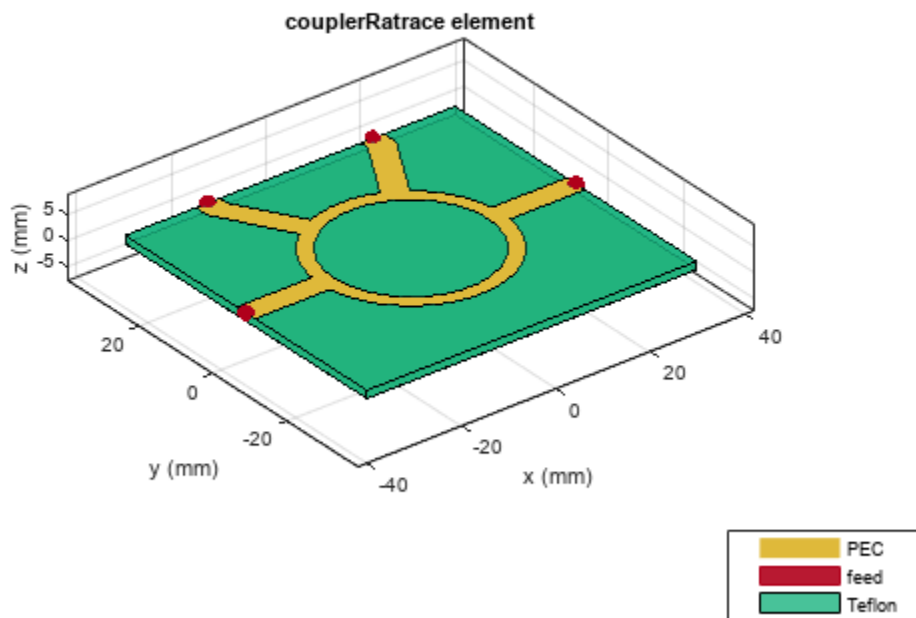
`layout(rfpcbobject)` displays all the metal layers and the PCB shape in the figure window. The red filled circle correspond to PCB feed points and the blue filled circles correspond to vias.

Examples

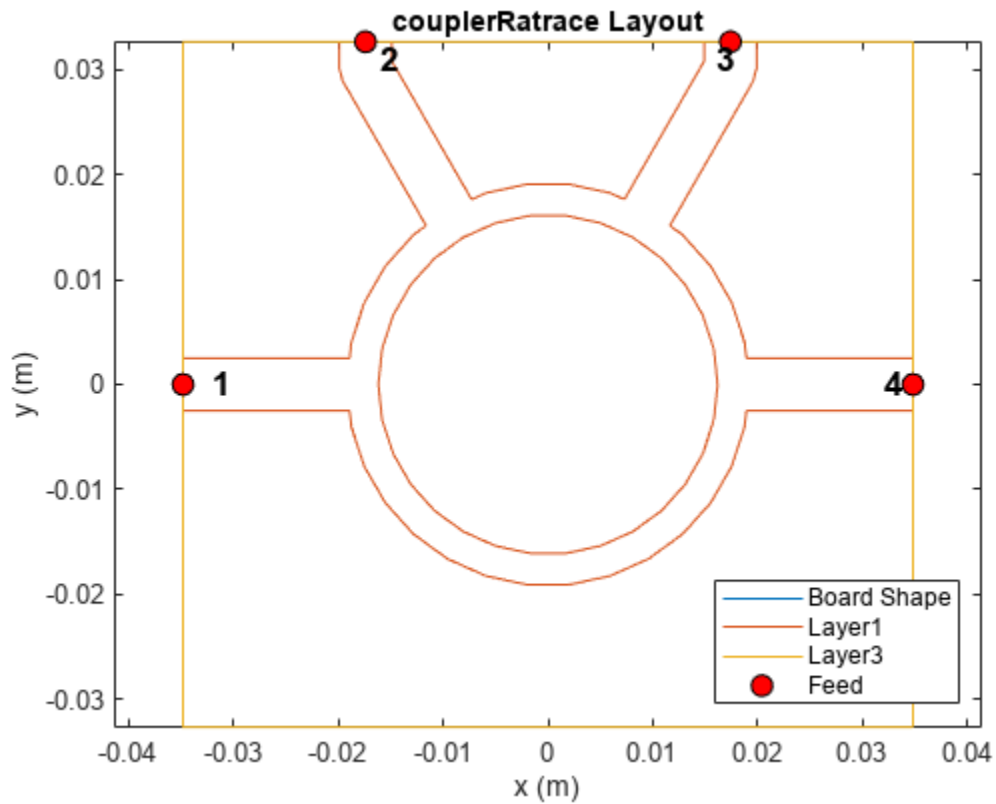
Display Ratrace Coupler Layout

Display the layout of a ratrace coupler.

```
coupler = couplerRatrace;  
show(coupler)
```



```
layout(coupler)
```



Input Arguments

rfpcbobject – PCB component

object handle

PCB component, specified as an object handle. For complete list of PCB components and shapes, see “PCB Components Catalog”

Version History

Introduced in R2021b

See Also

[show](#) | [info](#)

shapes

Extract all metal layer shapes of PCB component

Syntax

```
shapes(rfpcbobject)
```

Description

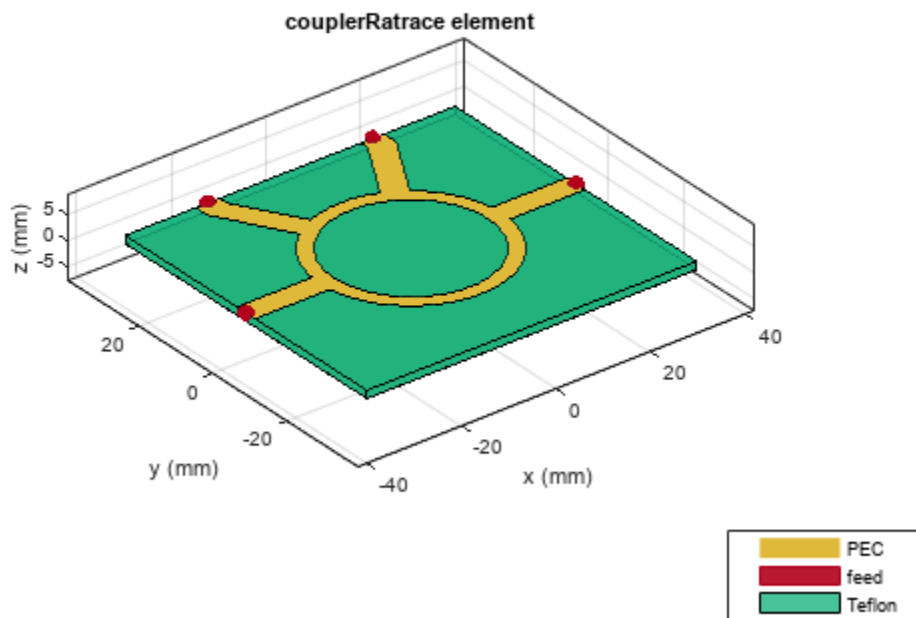
`shapes(rfpcbobject)` extracts all metal layer shapes of a PCB component and organizes them into an output structure.

Examples

Extract Rat-race Coupler Shapes

Extract the shapes of a rat-race coupler.

```
coupler = couplerRatrace;  
show(coupler)
```



```
s = shapes(coupler)

s = struct with fields:
  Layer1: [1x1 antenna.Polygon]
  Layer2: [1x1 antenna.Rectangle]
```

Input Arguments

rfpcbobject – PCB component object

RF PCB object

PCB component object, specified as an RF PCB object. For a complete list of the PCB components, see “PCB Components Catalog”.

Version History

Introduced in R2021b

See Also

layout | show

design

Design Wilkinson splitter around specified frequency

Syntax

```
wsplitter = design(wsplitterobj,frequency)
wsplitter = design( ____,Name=Value)
```

Description

`wsplitter = design(wsplitterobj,frequency)` designs a Wilkinson splitter around the specified frequency.

`wsplitter = design(____,Name=Value)` designs a Wilkinson splitter with additional options specified using name-value arguments.

Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

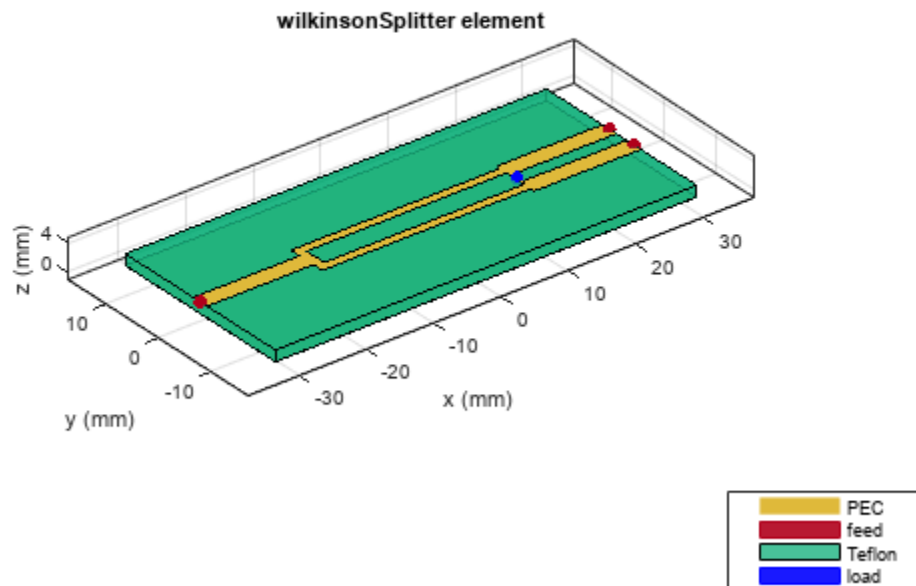
Design Wilkinson Splitter at 1.8 GHz

Design a Wilkinson splitter at 1.8 GHz and with a Z_0 of 75 ohm.

```
wsplitter = design(wilkinsonSplitter,1.8e9,Z0=75);
```

View the splitter.

```
figure;
show(wsplitter);
```

Input Arguments

wsplitterobj — Wilkinson splitter

`wilkinsonSplitter` object

Wilkinson splitter, specified as a `wilkinsonSplitter` object.

Example: `wsplitterobj = wilkinsonSplitter; design(wsplitterobj,2e9)` designs a Wilkinson splitter around a frequency of 2 GHz.

frequency — Design frequency of Wilkinson splitter

real positive scalar

Design frequency of the Wilkinson splitter, specified as a real positive scalar in hertz.

Example: `55e6`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=2`

Z0 – Characteristic impedance of splitter

50 (default) | positive scalar

Characteristic impedance of the splitter, specified as a positive scalar in ohms.

Data Types: double

Output Arguments

wsplitter – Wilkinson splitter operating around specified frequency

wilkinsonSplitter object

Wilkinson splitter operating around the specified frequency, returned as a wilkinsonSplitter object.

Version History

Introduced in R2021b

See Also

sparameters

design

Design unequal Wilkinson splitter around specified frequency

Syntax

```
uwsplitter = design(uwsplitterobj,frequency)
uwsplitter = design( ___,Name=Value)
```

Description

`uwsplitter = design(uwsplitterobj,frequency)` designs a unequal Wilkinson splitter around the specified frequency.

`uwsplitter = design(___,Name=Value)` designs a unequal Wilkinson splitter with additional options specified using name-value arguments.

Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

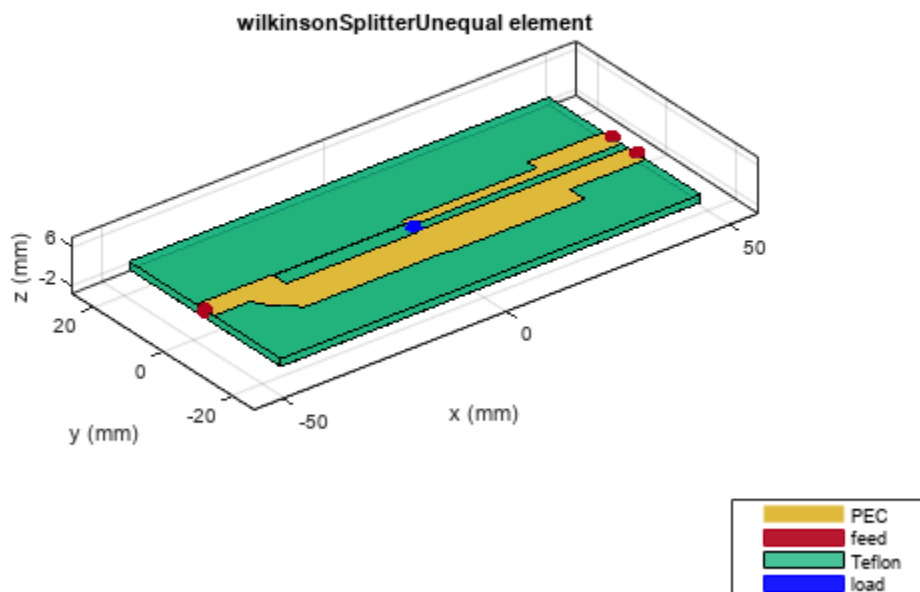
Design Unequal Wilkinson Splitter at 1.8 GHZ

Design an unequal wilkinson at 1.8 GHz with a characteristic impedance of 50 ohms and a PowerRatio of 6.

```
uwilk = design(wilkinsonSplitterUnequal,1.8e9,Z0=50,PowerRatio=6);
```

View the splitter

```
show(uwilk);
```



Input Arguments

uwsplitterobj — Unequal Wilkinson splitter

`wilkinsonSplitterUnequal` object

Unequal Wilkinson splitter, specified as a `wilkinsonSplitterUnequal` object.

Example: `uwsplitterobj = wilkinsonSplitterUnequal; design(uwsplitterobj,2e9)` designs a unequal Wilkinson splitter around a frequency of 2 GHz.

frequency — Design frequency of unequal Wilkinson splitter

real positive scalar

Design frequency of the unequal Wilkinson splitter, specified as a real positive scalar in hertz.

Example: `2.5e9`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=75`

Z0 – Characteristic impedance of splitter

50 (default) | positive scalar

Characteristic impedance of the splitter, specified as a positive scalar in ohms.

Data Types: double

PowerRatio – Power division ratio between two output ports of splitter

2 (default) | positive scalar

Power division ratio between the two output ports of the splitter, specified as a positive scalar.

Data Types: double

Output Arguments**uwsplitter – Unequal Wilkinson splitter operating around specified frequency**`wilkinsonSplitterUnequal` object

Unequal Wilkinson splitter operating around the specified frequency, returned as a `wilkinsonSplitterUnequal` object.

Version History**Introduced in R2021b****See Also**

design

Design coupled line filter around specified frequency

Syntax

```
clfilter = design(clfilterobj,frequency)
clfilter = design( ___,Name,Value)
```

Description

`clfilter = design(clfilterobj,frequency)` designs a coupled line filter around the specified frequency.

`clfilter = design(___,Name,Value)` designs a unequal Wilkinson splitter with additional options specified using name-value arguments.

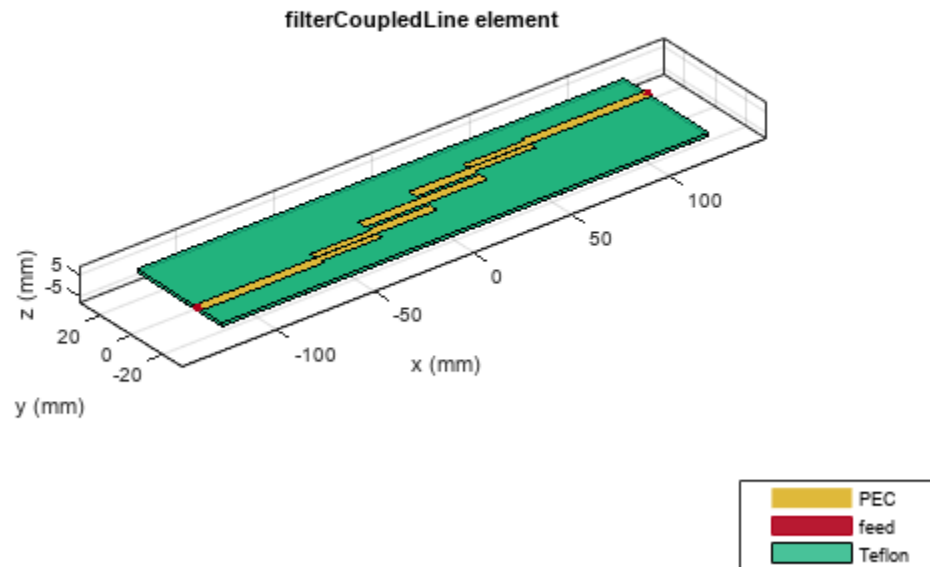
Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

Design Coupled Line Filter Around 1.8 GHz.

Design a coupled line filter around 1.8 GHz and a fractional bandwidth of 10 percent.

```
clfilter = design(filterCoupledLine,1.8e9,FBW=10);
show(clfilter)
```



Input Arguments

clfilterobj – Coupled line filter

`filterCoupledLine` object

Coupled line filter, specified as a `filterCoupledLine` object.

Example: `clfilterobj = filterCoupledLine; design(clfilterobj,2e9)` designs a coupled line filter around a frequency of 2 GHz.

frequency – Design frequency of coupled line filter

real positive scalar

Design frequency of coupled line filter, specified as a real positive scalar in hertz.

Example: `5e9`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `FilterType='ButterWorth'`

FilterType — Type of filter

'Butterworth' (default) | positive scalar

Type of filter, specified as 'Butterworth', 'Chebyshev', or 'InverseChebyshev'.

Data Types: double

FBW — Fractional bandwidth of filter response

10 (default) | positive scalar

Fractional bandwidth of the filter response, specified as positive scalar in percents.

Data Types: double

RippleFactor — Passband factor of Chebyshev filter

0.01 (default) | positive scalar

Passband factor of the Chebyshev filter, specified as positive scalar in decibels. For Butterworth filter, the passband factor is not required.

Data Types: double

Output Arguments**clfilter — Coupled line filter around specified reference frequency**

filterCoupledLine object

Coupled line filter around the specified frequency, returned as a filterCoupledLine object.

Version History

Introduced in R2021b

See Also

sparameters

design

Design stepped impedance low pass filter around desired cut-off frequency

Syntax

```
sifilter = design(sifilterobj,frequency)
sifilter = design( ____,Name=Value)
```

Description

`sifilter = design(sifilterobj,frequency)` designs a stepped impedance low pass filter around the cut-off frequency.

`sifilter = design(____,Name=Value)` designs a stepped impedance low pass filter with additional options specified by name-value arguments.

Note

- PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.
 - The design for stepped impedance low pass filter is based on analytical equations. Analyzing the parameters using EM-simulation model causes a shift in the cut-off frequency towards the lower end of the frequency range. This is an expected behavior due to the coupling effect.
-

Examples

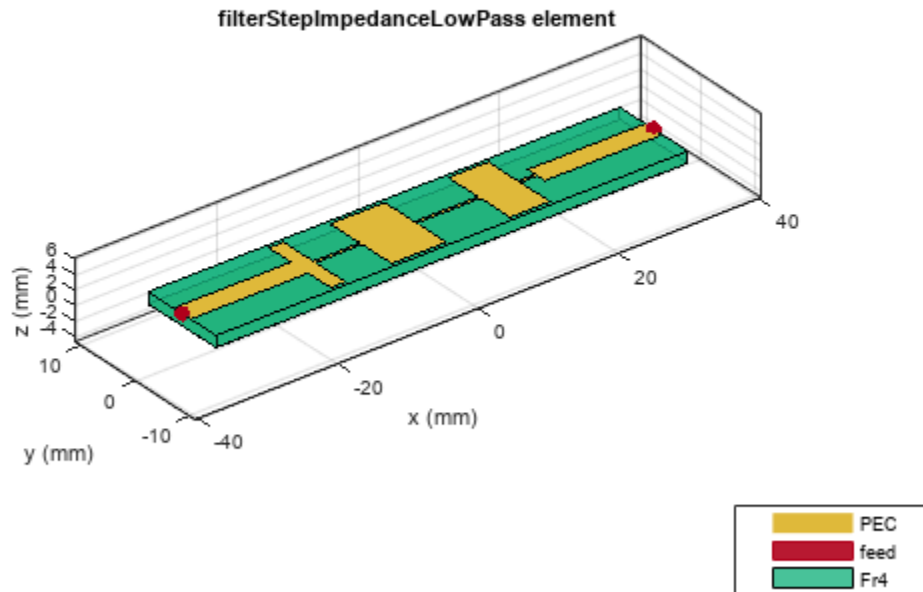
Design Stepped Impedance Low Pass Filter at 2.5 GHz

Design a sixth order stepped impedance low pass filter at 2.5 GHz with 20 ohm low impedance line, 120 ohm high impedance line on FR4 substrate of thickness 1.58 mm.

```
sifilter = filterStepImpedanceLowPass(FilterOrder=6,Height=1.58e-3) ;
Sub = dielectric(Name='Fr4',EpsilonR=4.2,LossTangent=0.02,Thickness=1.58e-3);
sifilter.Substrate = Sub;
sifilterobj = design(sifilter,2.5e9,Z0=50,LowZ=20,HighZ=120);
```

View the filter.

```
show(sifilterobj);
```



Input Arguments

sifilterobj — Stepped impedance low pass line filter

`filterStepImpedanceLowPass` object

Stepped impedance low pass filter, specified as a `filterStepImpedanceLowPass` object.

Example: `sifilterobj = filterStepImpedanceLowPass; design(sifilterobj,2e9)` designs a coupled line filter around a frequency of 2 GHz.

frequency — Design frequency of stepped impedance low pass filter

real positive scalar

Design frequency of the stepped impedance low pass filter, specified as a real positive scalar in hertz.

Example: `5e9`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=75`

Z0 – Reference impedance

50 (default) | positive scalar

Reference impedance, specified as a positive scalar in ohms.

Data Types: double

LowZ – Low impedance line

20 (default) | positive scalar

Low impedance line, specified as a positive scalar in ohms.

Data Types: double

HighZ – High impedance line

120 (default) | positive scalar

High impedance line, specified as a positive scalar in ohms.

Data Types: double

FilterType – Type of filter

'Butterworth' (default) | positive scalar

Type of filter, specified as 'Butterworth', or 'Chebyshev'.

Data Types: char | string

RippleFactor – Passband factor of Chebyshev filter

0.01 (default) | positive scalar

Passband factor of the Chebyshev filter, specified as positive scalar in decibels. For Butterworth filter, the passband factor is not required.

Data Types: double

Output Arguments**sifilter – Stepped impedance low pass filter around specified frequency**

filterStepImpedanceLowPass object

Stepped impedance low pass filter around specified frequency, returned as a filterStepImpedanceLowPass object.

Version History

Introduced in R2021b

See Also

sparameters

design

Design hairpin filter around specified frequency

Syntax

```
hpfilter = design(hpfilterobj,frequency)
hpfilter = design( ____,Name=Value)
```

Description

`hpfilter = design(hpfilterobj,frequency)` designs a hairpin filter around the specified frequency.

`hpfilter = design(____,Name=Value)` designs a hairpin filter with additional options specified by name-value arguments.

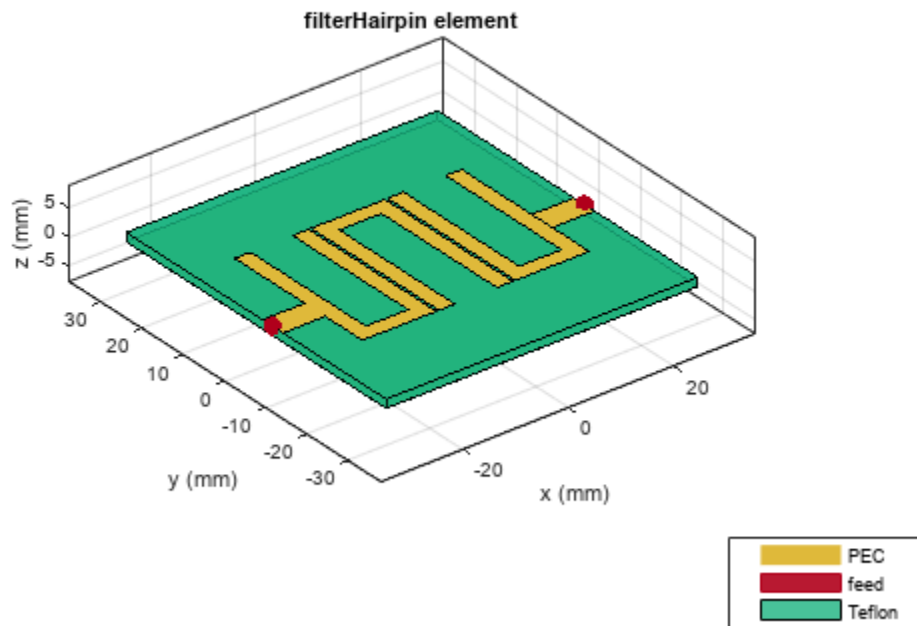
Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

Design Chebyshev Hairpin Filter at 1.8 GHz

Design a Hairpin filter with a Chebyshev response at 1.8 GHz and a fractional bandwidth of 10 percent.

```
hpfilt = design(filterHairpin,1.8e9,FBW=10,FilterType='Chebyshev');
show(hpfilt);
```



Design Fifth Order Hairpin Filter

Design a 5th order tapped input hairpin filter with a Chebyshev response at 1.8 GHz and a fractional bandwidth of 10 percent.

```
hpfilt = filterHairpin(FeedType='Tapped')
```

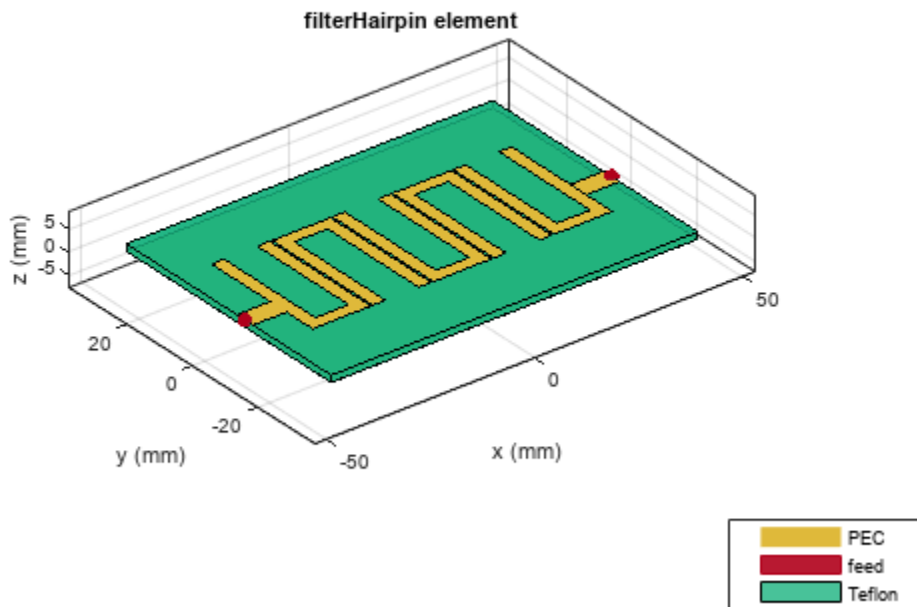
```
hpfilt =
  filterHairpin with properties:

    Resonator: [1x1 ubendRightAngle]
    FilterOrder: 3
    ResonatorOffset: [0 0 0]
    Spacing: [4.0000e-04 4.0000e-04]
    PortLineLength: 0.0080
    PortLineWidth: 0.0050
    FeedOffset: [-0.0055 -0.0055]
    FeedType: 'Tapped'
    Height: 0.0016
    GroundPlaneWidth: 0.0567
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

```
hpfilt.FilterOrder = 5;  
hpfilt = design(hpfilt, 1.8e9,FBW=10,FilterType='Chebyshev');
```

View the filter.

```
show(hpfilt);
```



Input Arguments

hpfilterobj — Hairpin filter

filterHairpin object

Hairpin filter, specified as a filterHairpin object.

Example: `hpfilterobj = filterHairpin; design(hpfilterobj,2e9)` designs a hairpin filter around a frequency of 2 GHz.

frequency — Design frequency of hairpin filter

real positive scalar

Design frequency of the hairpin filter, specified as a real positive scalar in hertz.

Example: 5e9

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `RippleFactor=0.02`

FilterType — Type of filter

'Butterworth' (default) | positive scalar

Type of filter, specified as 'Butterworth', or 'Chebyshev'.

Data Types: `char` | `string`

RippleFactor — Passband factor of Chebyshev filter

0.01 (default) | positive scalar

Passband factor of the Chebyshev filter, specified as positive scalar in decibels. For Butterworth filter, the passband factor is not required.

Data Types: `double`

Output Arguments

hpfilter — Hairpin filter operating around specified frequency

`filterHairpin` object

Hairpin filter operating around the specified frequency, returned as a `filterHairpin` object.

Version History

Introduced in R2021b

See Also

`sparameters`

pcbcascade

Create new component using cascade operation

Syntax

```
combinedcomponent = pcbcascade(component1,component2)
combinedcomponent = pcbcascade(component1,component2,m,n)
```

Description

`combinedcomponent = pcbcascade(component1,component2)` creates a new component by using a cascade operation along port 2 of the first component and port 1 of the second component.

`combinedcomponent = pcbcascade(component1,component2,m,n)` creates a new component by using a cascade operation along port *m* of the first component and port *n* of the second component.

Note

- `pcbcascade` only supports: 2-metal layer PCB components, feeds specified at the edge of components, and identical substrate properties in both components.
 - If either of the components is an antenna, the new component that the object creates is a `pcbStack` object.
 - To use `pcbStack` object you require Antenna Toolbox.
-

Examples

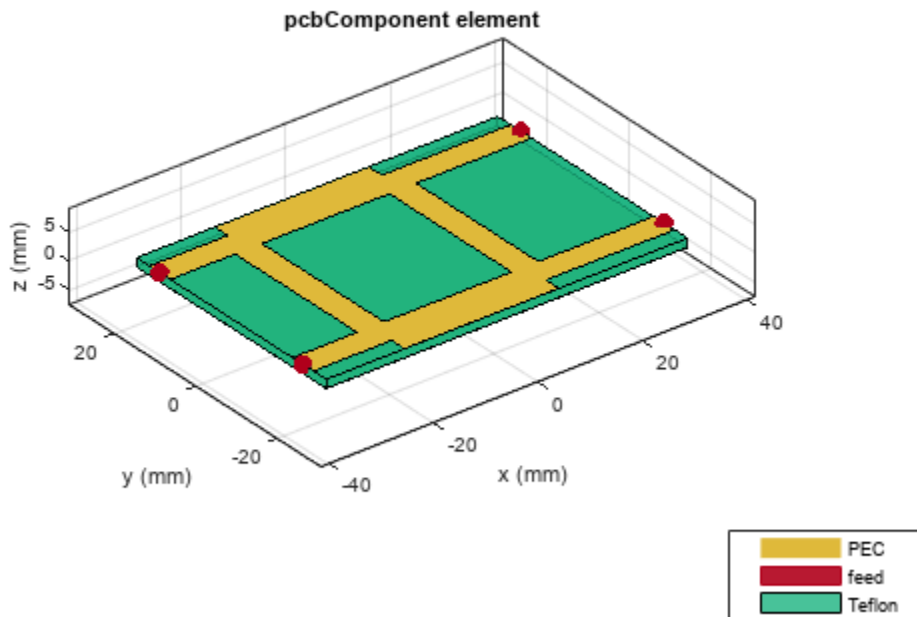
Create Component From Branchline Coupler and Coupled Microstrip Line

Create a new component by cascading a branchline coupler with a coupled microstrip line.

```
c = design(couplerBranchline,5.6e9);
mc = design(coupledMicrostripLine,5.6e9);
mc.Spacing = c.ShuntArmLength;
r = pcbcascade(c,mc);
```

View the new component.

```
figure
show(r)
```

Input Arguments

component1 — PCB component or antenna

PCB component object | antenna object

PCB component or antenna, specified a PCB component object or antenna object. For a complete list of the PCB components, see “PCB Components Catalog”.

Example: `mline1 = microstripLine; mline2 = design(microstripLine,3e9); component = pcbcascade(mline1,mline2)` creates a new component by cascading `mline1` and `mline2`.

Data Types: char | string

component2 — PCB component or antenna

PCB component object | antenna object

PCB component or antenna, specified a PCB component object or antenna object. For a complete list of the PCB components, see “PCB Components Catalog”.

Example: `mline1 = microstripLine; mline2 = design(microstripLine,3e9); component = pcbcascade(mline1,mline2)` creates a new component by cascading `mline1` and `mline2`.

Data Types: char | string

m – Port number of first component

1 (default) | positive scalar

Port number of the first component, specified as a positive scalar.

Example: `coupler = couplerRatrace;mline = microstripLine;component = pcbcascade(coupler,mline,3,1)` creates a new component by cascading Port 3 of the coupler to Port 1 of the microstrip transmission line.

Data Types: double

n – Port number of second component

1 (default) | positive scalar

Port number of second component, specified as a positive scalar.

Example: `coupler = couplerRatrace;mline = microstripLine;component = pcbcascade(coupler,mline,3,1)` creates a new component by cascading Port 3 of the coupler to Port 1 of the microstrip transmission line.

Data Types: double

Version History

Introduced in R2021b

See Also

`pcbComponent`

design

Design PCB component around particular frequency

Syntax

```
rfpcbcomponent = design(rfpcbobject,frequency)
```

Description

`rfpcbcomponent = design(rfpcbobject,frequency)` designs a PCB component around the specified frequency.

Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

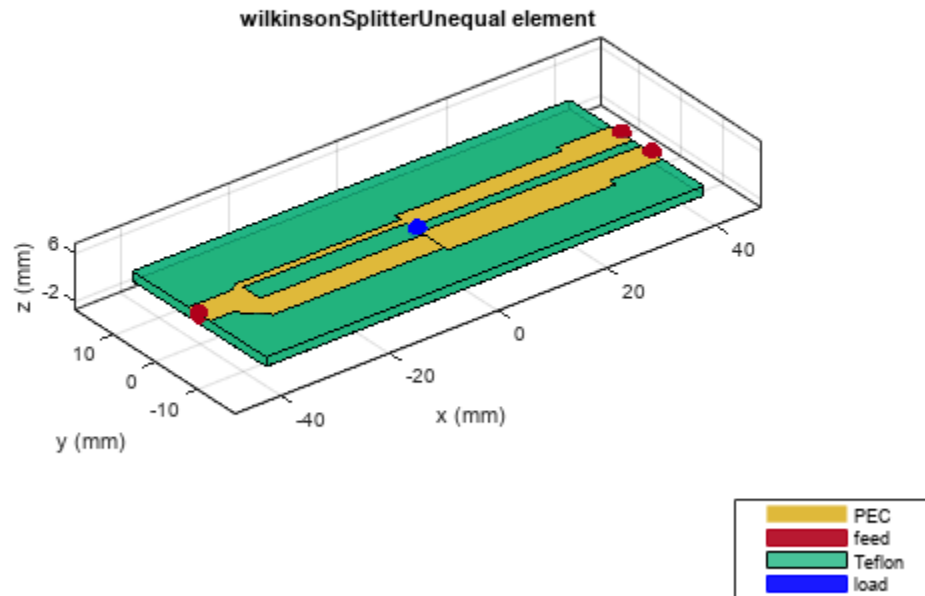
Design Unequal Wilkinson Splitter at 1.8 GHZ

Design an unequal Wilkinson around 1.8 GHz.

```
uwilk = design(wilkinsonSplitterUnequal,1.8e9);
```

View the splitter

```
show(uwilk);
```



Plot S-parameters between the frequency range of 0.1 GHz to 6 GHz.

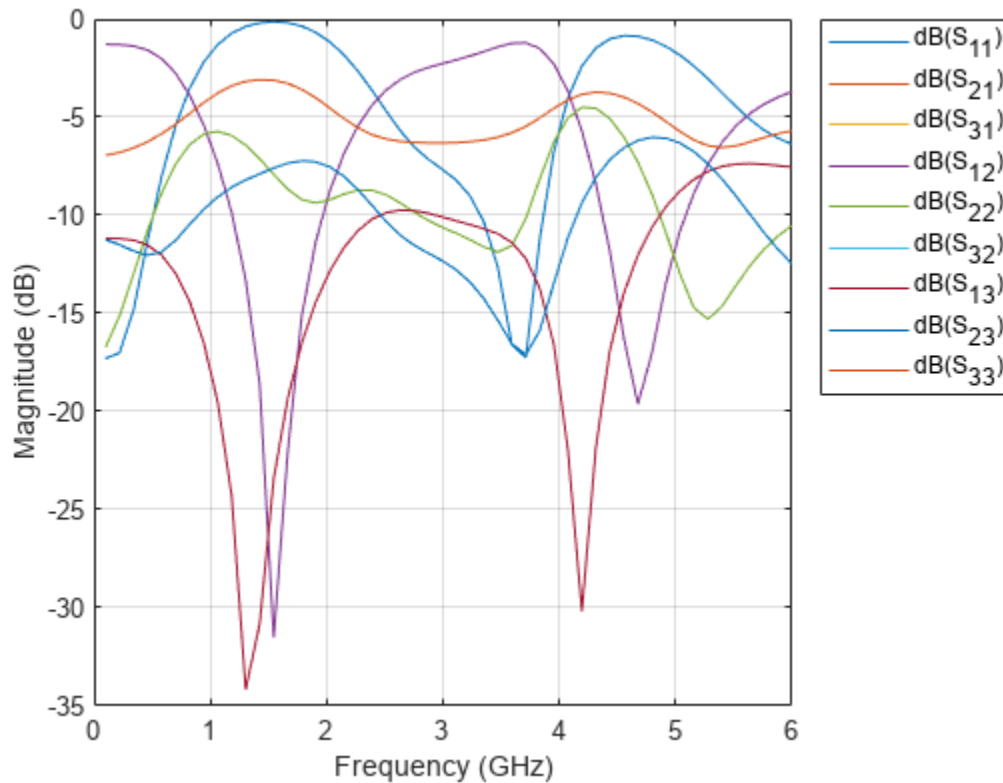
```
spar = sparameters(uwilk,linspace(0.1e9,6e9,50))
```

```
spar =  
  sparameters: S-parameters object
```

```
    NumPorts: 3  
    Frequencies: [50x1 double]  
    Parameters: [3x3x50 double]  
    Impedance: 50
```

```
rfparam(obj,i,j) returns S-parameter  $S_{ij}$ 
```

```
rfplot(spar)
```



Input Arguments

rpfcbobject – PCB component object

object handle

PCB component object, specified as a RF PCB object. For complete list of PCB components, microstrip bends, and traces, see “PCB Components Catalog”.

Note The following PCB catalog components are not supported by this function: `spiralInductor`, `interdigitalCapacitor`, and `stubRadialShunt`.

frequency – Design frequency of PCB component

real positive scalar

Design frequency of the PCB component, specified as a real positive scalar in hertz.

Example: `3e9`

Data Types: `double`

Output Arguments

rfpcbcomponent — PCB catalog component operating around specified frequency
object handle

PCB catalog component operating around specified frequency, returned as a object handle.

Version History

Introduced in R2021b

See Also

sparameters

mirrorX

Mirror shape along X-axis

Syntax

```
mirroredshape = mirrorX(shape)
```

Description

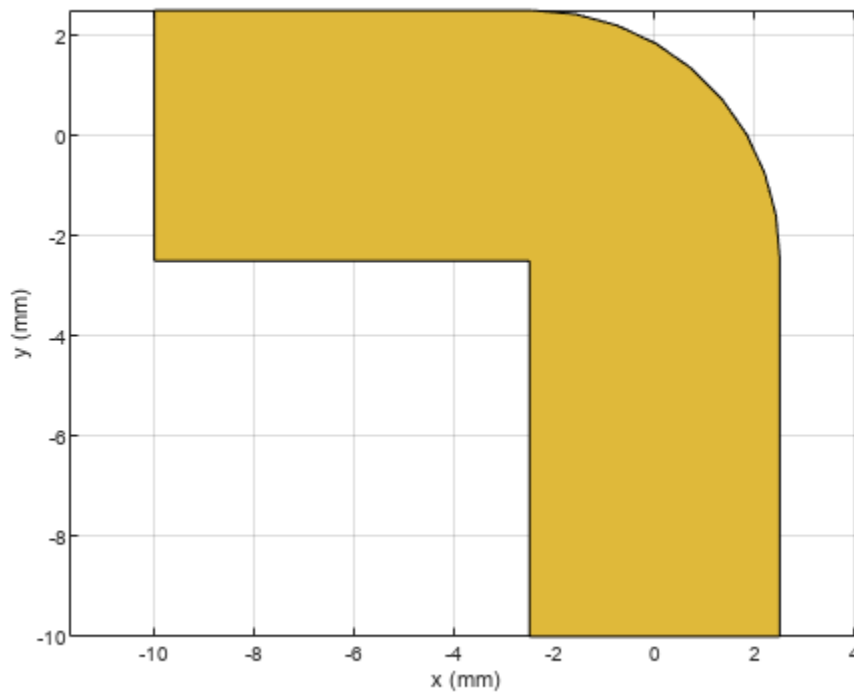
`mirroredshape = mirrorX(shape)` mirrors a shape along the X-axis.

Examples

Mirror Curved Bend Shape Along X-Axis

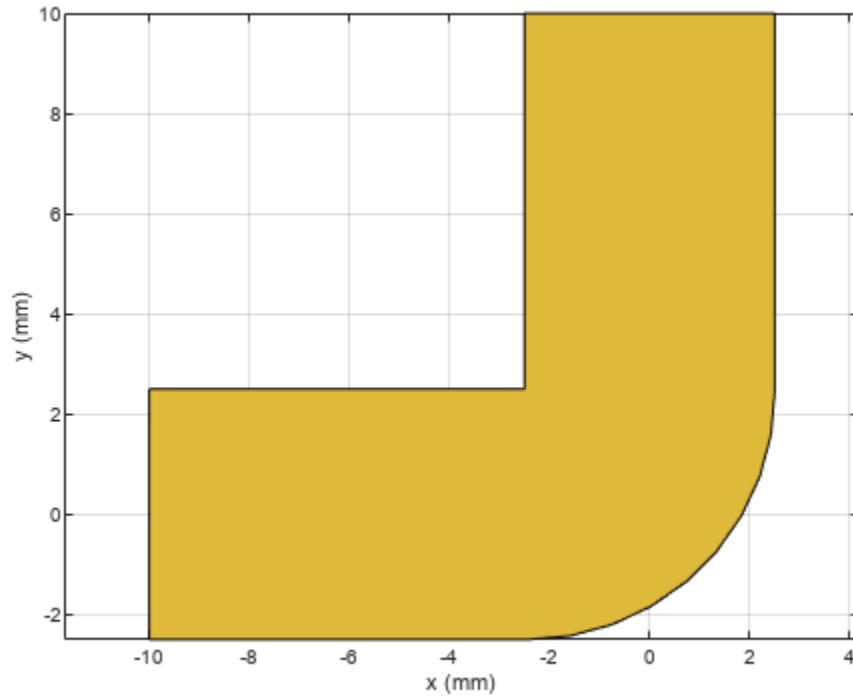
Create a curved bend and view it.

```
shape = bendCurved;  
show(bendCurved)
```



Mirror the shape along the X-axis.

```
mirrorX(shape)
```



Input Arguments

shape – Shape to mirror

shape or PCB object

Shape to mirror, specified as `shape` or PCB object You can specify any of the shapes in “Custom Geometry and PCB Fabrication”.

Example: `shape = bendCurved`; creates a `bendCurved` shape object.

Version History

Introduced in R2022a

See Also

`add` | `subtract` | `area` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot`

mirrorY

Mirror shape along Y-axis

Syntax

```
mirroredshape = mirrorY(shape)
```

Description

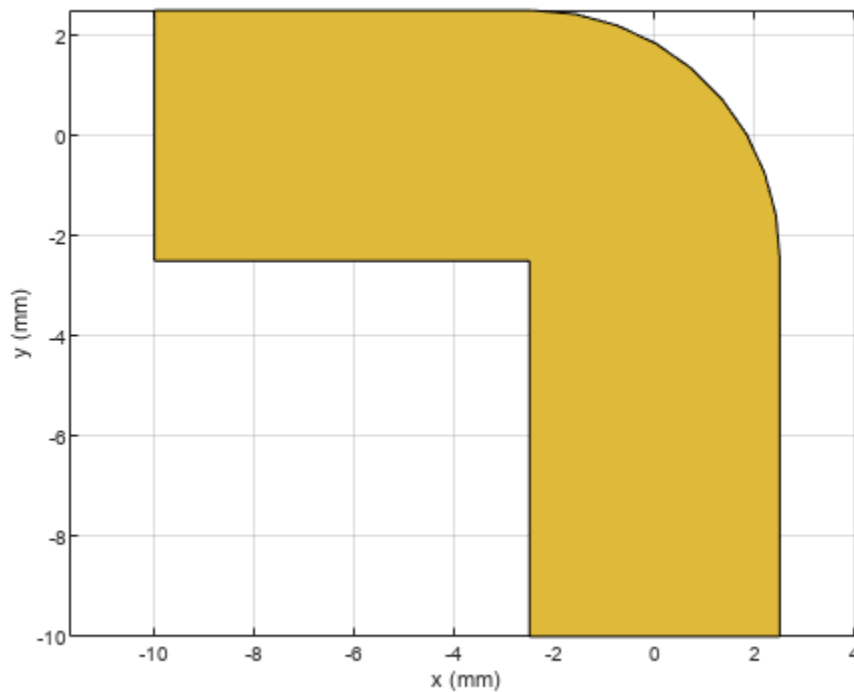
`mirroredshape = mirrorY(shape)` mirrors a shape along the Y-axis.

Examples

Mirror Curved Bend Shape Along Y-Axis

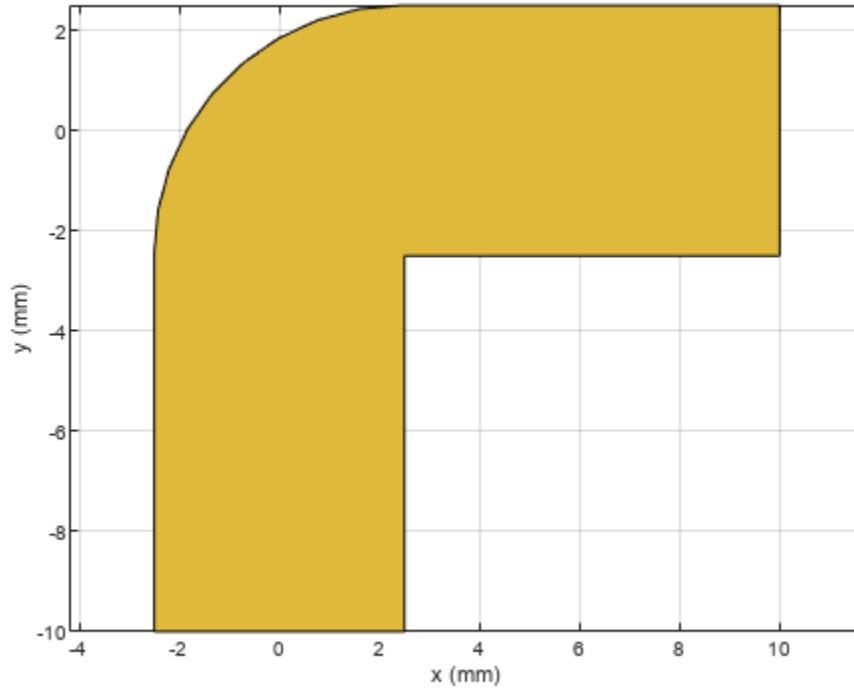
Create a curved bend and view it.

```
shape = bendCurved;  
show(bendCurved)
```



Mirror the shape along the Y-axis.

`mirrorY(shape)`



Input Arguments

shape – Shape to mirror

shape or PCB object

Shape to mirror, specified as `shape` or PCB object You can specify any of the shapes in “Custom Geometry and PCB Fabrication”.

Example: `shape = bendCurved`; creates a `bendCurved` shape object.

Version History

Introduced in R2022a

See Also

`add` | `subtract` | `area` | `rotate` | `rotateX` | `rotateY` | `rotateZ` | `translate` | `show` | `mesh` | `plot`

coupling

Calculate coupling factor of coupler

Syntax

```
coupling(coupler, frequency)
c = coupling(coupler, frequency)
```

Description

`coupling(coupler, frequency)` calculates and plots the coupling factor of a coupler over the specified frequency values.

`c = coupling(coupler, frequency)` returns the coupling factor of a coupler over the specified frequency.

Examples

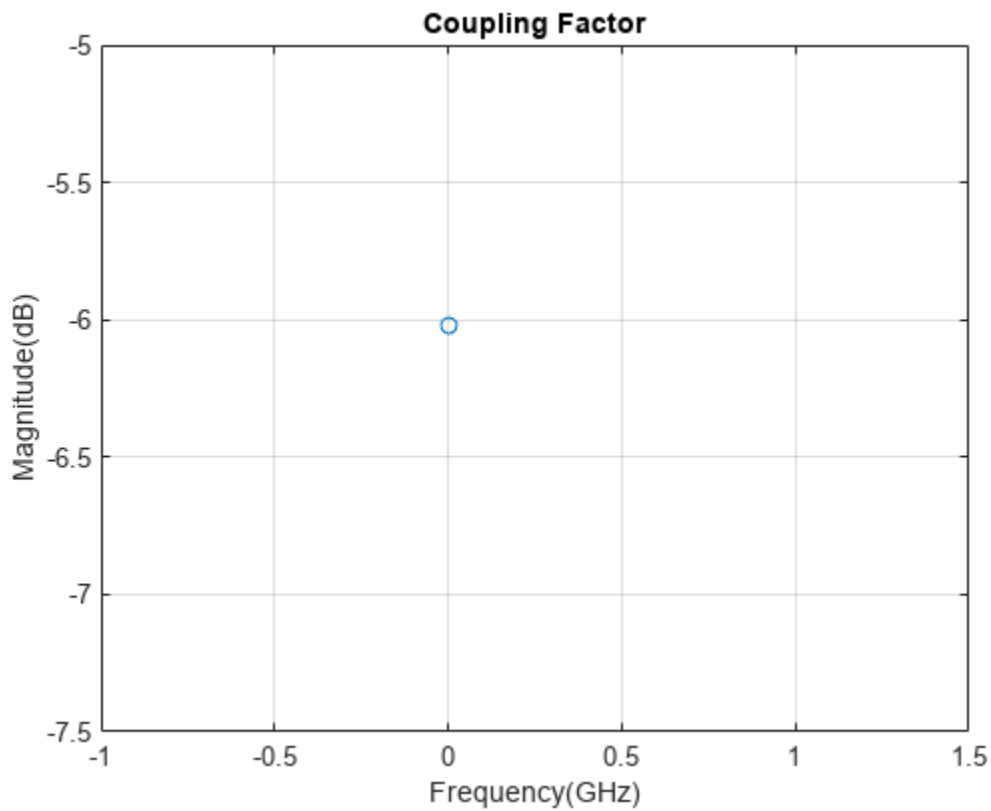
Coupling Factor of Branchline Coupler

Create a branchline coupler with default values.

```
coupler = couplerBranchline
coupler =
  couplerBranchline with properties:
    PortLineLength: 0.0186
    PortLineWidth: 0.0051
    SeriesArmLength: 0.0184
    SeriesArmWidth: 0.0083
    ShuntArmLength: 0.0186
    ShuntArmWidth: 0.0051
    Height: 0.0016
    GroundPlaneWidth: 0.0600
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

Calculate the coupling factor of the coupler at 2 GHz.

```
coupling(coupler, 2e6)
```



Coupling Factor of Rat-Race Coupler

Create a rat-race coupler with default values.

```
coupler = couplerRatrace;
```

Calculate the coupling factor of the coupler at 3 GHz.

```
c = coupling(coupler,3e6)
```

```
c = -6.0206
```

Input Arguments

coupler — Coupler

coupler object

Coupler, specified as a coupler object. For a complete list of couplers, see “Splitters and Couplers”.

frequency — Frequency to calculate coupling

scalar | vector

Frequency to calculate the coupling, specified as an integer in Hz or as a vector with each element specified in Hz.

Version History

Introduced in R2022a

See Also

directivity | isolation

directivity

Calculate directivity of coupler

Syntax

```
directivity(coupler, frequency)
d = directivity(coupler, frequency)
```

Description

`directivity(coupler, frequency)` calculates and plots the directivity of a coupler over the specified frequency values.

`d = directivity(coupler, frequency)` returns the directivity of a coupler over the specified frequency.

Examples

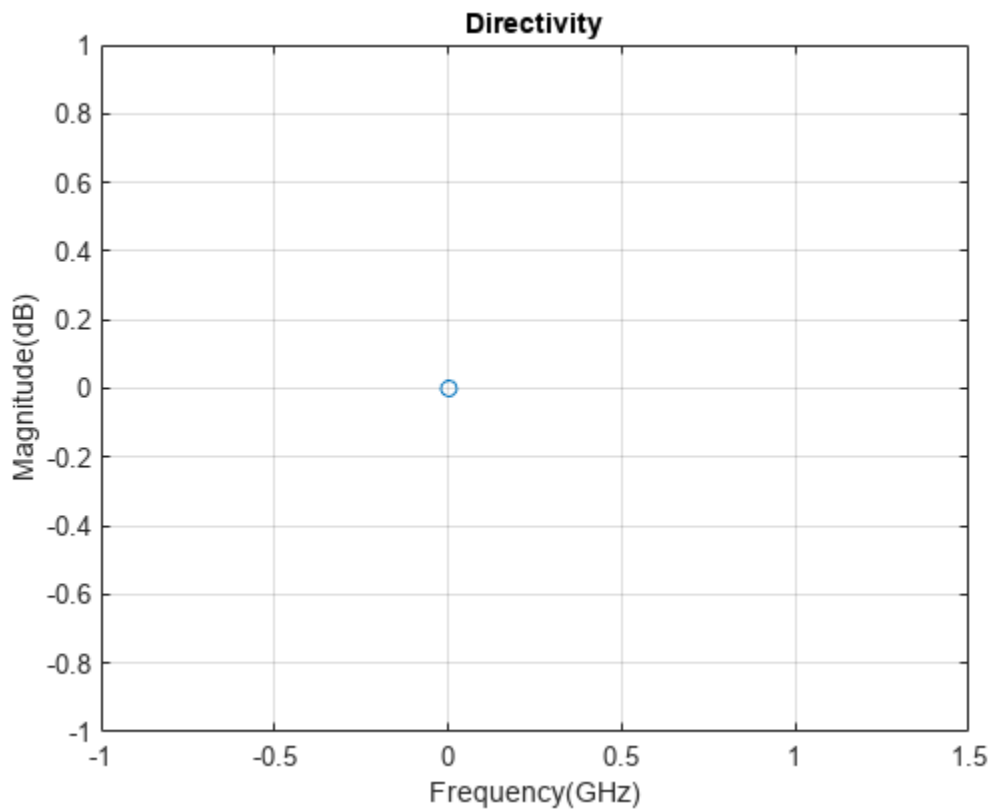
Directivity of Branchline Coupler

Create a branchline coupler with default values.

```
coupler = couplerBranchline
coupler =
    couplerBranchline with properties:
        PortLineLength: 0.0186
        PortLineWidth: 0.0051
        SeriesArmLength: 0.0184
        SeriesArmWidth: 0.0083
        ShuntArmLength: 0.0186
        ShuntArmWidth: 0.0051
        Height: 0.0016
        GroundPlaneWidth: 0.0600
        Substrate: [1x1 dielectric]
        Conductor: [1x1 metal]
```

Calculate the directivity of the coupler at 2 GHz.

```
directivity(coupler, 2e6)
```



Directivity of Rat-Race Coupler

Create a rat-race coupler with default values.

```
coupler = couplerRatrace;
```

Calculate the directivity of the coupler at 3 GHz.

```
c = directivity(coupler,3e6)
```

```
c = -7.3107e-06
```

Input Arguments

coupler — Coupler

coupler object

Coupler, specified as a coupler object. For a complete list of couplers, see “Splitters and Couplers”.

frequency — Frequency to calculate directivity

scalar | vector

Frequency to calculate the directivity, specified as an integer in Hz or as a vector with each element specified in Hz.

Version History

Introduced in R2022a

See Also

`coupling` | `isolation`

isolation

Calculate isolation of coupler

Syntax

```
isolation(coupler, frequency)
i = isolation(coupler, frequency)
```

Description

`isolation(coupler, frequency)` calculates and plots the isolation of a coupler over the specified frequency values.

`i = isolation(coupler, frequency)` returns the isolation of a coupler over the specified frequency.

Examples

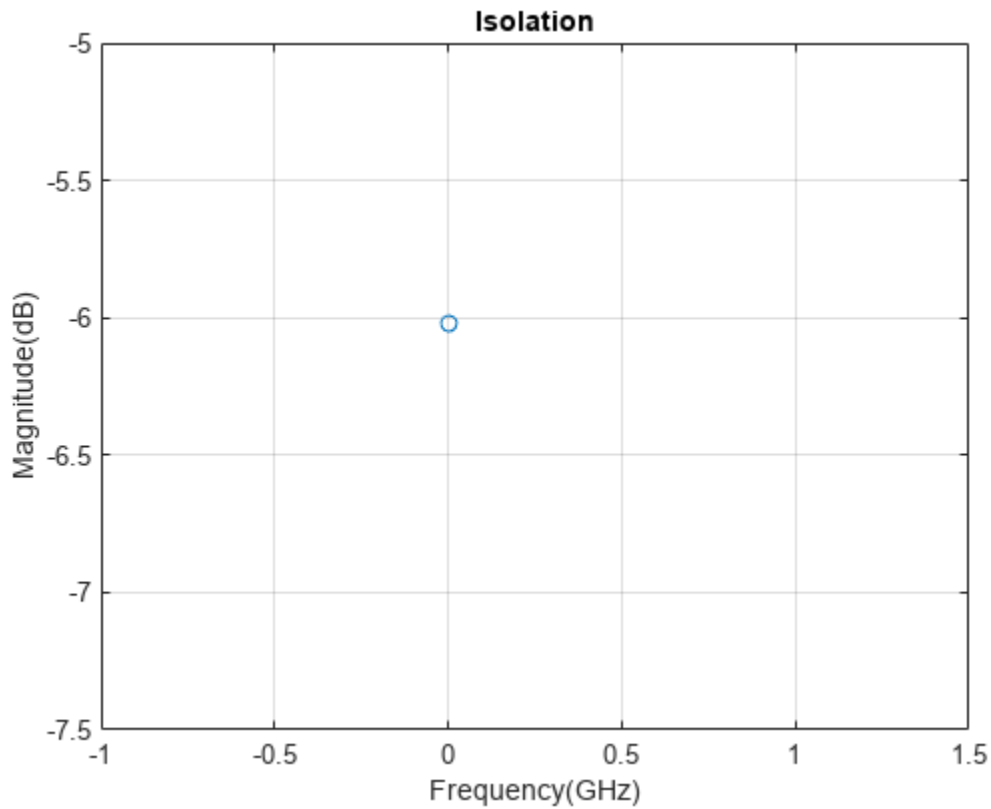
Isolation of Branchline Coupler

Create a branchline coupler with default values.

```
coupler = couplerBranchline
coupler =
  couplerBranchline with properties:
    PortLineLength: 0.0186
    PortLineWidth: 0.0051
    SeriesArmLength: 0.0184
    SeriesArmWidth: 0.0083
    ShuntArmLength: 0.0186
    ShuntArmWidth: 0.0051
    Height: 0.0016
    GroundPlaneWidth: 0.0600
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

Calculate the isolation of the coupler at 2 GHz.

```
isolation(coupler, 2e6)
```



Isolation of Rat-Race Coupler

Create a rat-race coupler with default values.

```
coupler = couplerRatrace;
```

Calculate the isolation of the coupler at 3 GHz.

```
i = isolation(coupler,3e6)
```

```
i = -6.0206
```

Input Arguments

coupler — Coupler

coupler object

Coupler, specified as a coupler object. For a complete list of couplers, see “Splitters and Couplers”.

frequency — Frequency to calculate isolation

scalar | vector

Frequency to calculate the isolation, specified as an integer in Hz or as a vector with each element specified in Hz.

Version History

Introduced in R2022a

See Also

coupling | directivity

design

Design corporate power divider around specified frequency

Syntax

```
divider = design(dividerobj,frequency)
divider = design( ____,Name=Value)
```

Description

`divider = design(dividerobj,frequency)` designs a corporate power divider around a specified frequency.

`divider = design(____,Name=Value)` designs a corporate power divider splitter with additional options specified using name-value arguments.

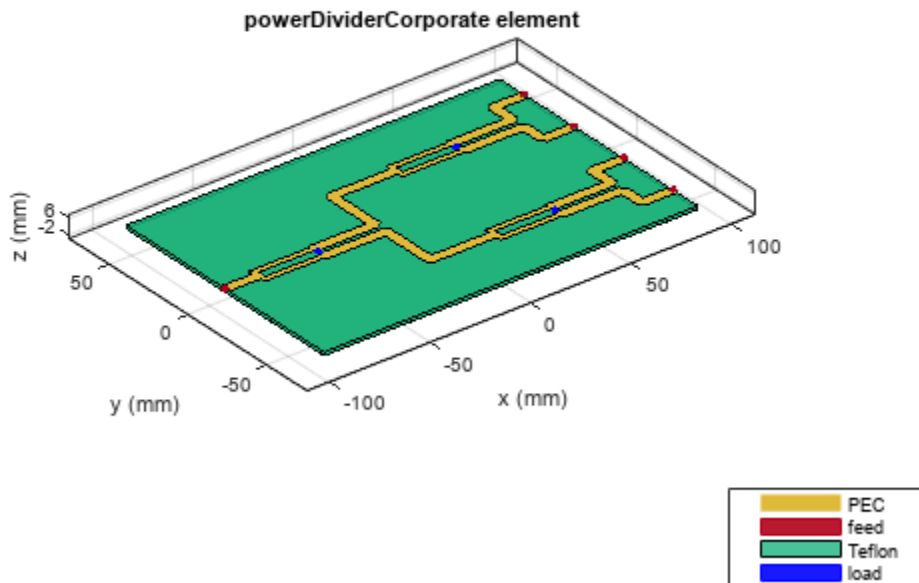
Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

1.8 GHz Corporate Power Divider

Design a corporate power divider at 1.8 GHz frequency and 50 ohms impedance.

```
pd = design(powerDividerCorporate,1.8e9,Z0=50);
figure;
show(pd);
```



Input Arguments

dividerobj — Corporate power divider

powerDividerCorporate object

Corporate power divider, specified as a powerDividerCorporate object.

Example: `dividerobj = wilkinsonSplitter; design(dividerobj,2e9)` designs a corporate power divider around a frequency of 2 GHz.

frequency — Design frequency of corporate power divider

real positive scalar

Design frequency of the corporate power divider, specified as a real positive scalar in hertz.

Example: `55e6`

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=2`

Z0 – Characteristic impedance of power divider

50 (default) | positive scalar

Characteristic impedance of the power divider, specified as a positive scalar in ohms.

Data Types: double

Output Arguments

dividerobj – Corporate power divider operating around specified frequency

powerDividerCorporate object

Corporate power divider operating around the specified frequency, returned as a powerDividerCorporate object.

Version History

Introduced in R2022a

See Also

sparameters

designCoupledLine

Calculate dimensions of coupled-line section for specified frequency

Syntax

```
[Length,Width,Spacing] = designCoupledLine(balunobj,frequency)
___ = designCoupledLine( ___,Name=Value)
```

Description

[Length,Width,Spacing] = designCoupledLine(balunobj,frequency) calculates the dimensions of the coupled line section of a coupled-line balun around a specified frequency.

Note designCoupledLine is the first step in designing a coupled line balun. This function is succeeded by designUncoupledLine and designOutputLine as the second and third step, respectively.

___ = designCoupledLine(___,Name=Value) calculates the dimensions of the coupled-line section of a coupled-line balun with additional options specified using name-value arguments.

Note PCB components designed using the design function operate around the specified frequency with a 10-15% tolerance.

Examples

Design Coupled Line Balun at 4 GHz

Define the frequency at 4 GHz.

```
f = 4e9;
```

Create a coupled line balun object.

```
balun = balunCoupledLine
```

```
balun =
    balunCoupledLine with properties:
```

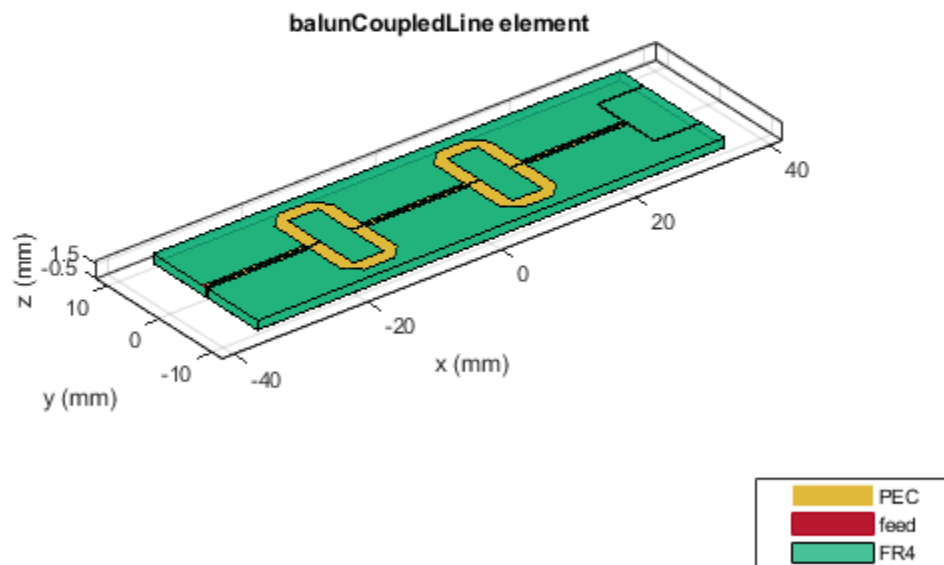
```
    NumCoupledLineSection: 3
    CoupledLineLength: 0.0153
    CoupledLineWidth: 4.0000e-04
    CoupledLineSpacing: 1.4000e-04
    UncoupledLineShape: [1x1 ubendMitered]
    OutputLineLength: 0.0124
    OutputLineWidth: 1.5300e-04
    OutputLineSpacing: 0.0110
    Height: 0.0013
```

```

GroundPlaneWidth: 0.0200
Substrate: [1x1 dielectric]
Conductor: [1x1 metal]

```

```
show(balun)
```



Step 1: Design coupled line section

Design the coupled line section of the balun with an even mode impedance of 159 ohms and an odd mode impedance of 51 ohms. Use the helper function **designCoupledLine**.

```
[ClineL,ClineW,ClineS] = designCoupledLine(balun,f,'Z0e',159,'Z0o',51)
```

```
ClineL = 0.0107
```

```
ClineW = 4.2682e-04
```

```
ClineS = 1.4374e-04
```

Step 2: Design uncoupled line section

Design the uncoupled line section of the balun with the even and odd mode impedance of 59 ohms. Use the helper function **designUncoupledLine**.

```
[unclineL,unclineW] = designUncoupledLine(balun,f,'Z0',59,'LineLength',0.25)
```

```
unclineL = 0.0103
```



```
unclineW = 0.0018
```

Step 3: Design output line section

Design the output line section of the balun at the same frequency to extend the port 2 and port3. Use the helper function **designOutputLine**.

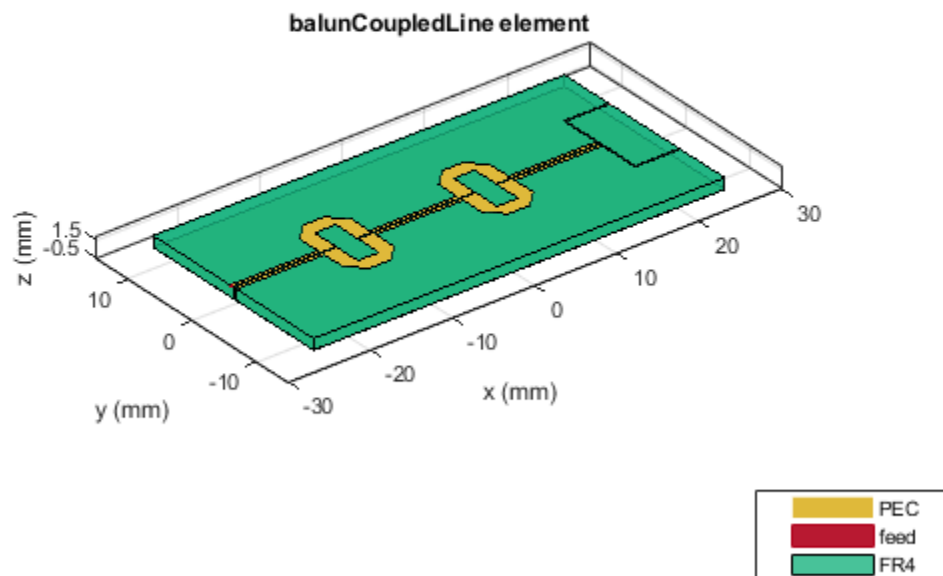
```
[OutL,OutW] = designOutputLine(balun,f,'Z0e',159,'Z0o',51,'Z0',59,'Zref',50)
```

```
OutL = 0.0109
```

```
OutW = 1.6115e-04
```

Set all the design dimensions to the coupled balun object.

```
balun.CoupledLineLength = ClineL;
balun.CoupledLineWidth = ClineW;
balun.CoupledLineSpacing = ClineS;
UnCoupledLine = ubendMitered;
UnCoupledLine.Length = [unclineL/2,unclineL/4,unclineL/2];
UnCoupledLine.Width = [unclineW,unclineW,unclineW];
balun.UnCoupledLineShape = UnCoupledLine;
balun.OutputLineLength = OutL;
balun.OutputLineWidth = OutW;
balun.OutputLineSpacing = OutL+ClineS;
gndW = 25e-3;
balun.GroundPlaneWidth = gndW;
show(balun)
```

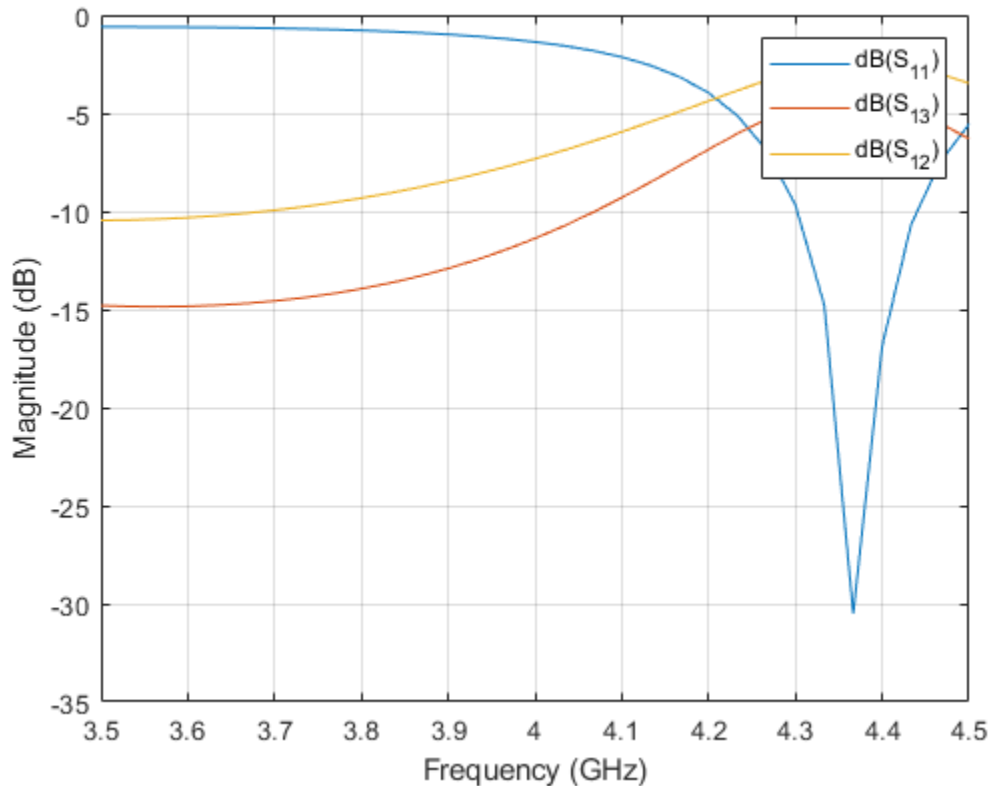


Analyze and plot the S-parameters of this balun.

```
s11 = sparameters(balun, linspace(3.5e9, 4.5e9, 31));
```

```
figure; rfplot(s11, 1, 1);
hold on; rfplot(s11, 1, 3)
hold on; rfplot(s11, 1, 2)
```

Click legend labels to toggle the line visibility



Input Arguments

balunobj — Coupled-line balun

balunCoupledLine object

Coupled-line balun, specified as a balunCoupledLine object.

Example: `balunobj = balunCoupledLine; design(balunobj, 2e9)` designs a coupled-line balun around a frequency of 2 GHz and returns the CoupledLineLength, CoupledLineWidth, and CoupledLineSpacing dimensions of the coupled line balun.

frequency — Design frequency of coupled-line balun

real positive scalar

Design frequency of coupled-line balun, specified as a real positive scalar in hertz.

Example: `55e6`

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0o=61`

Z0e — Even mode impedance

159 (default) | positive scalar

Even mode impedance, specified as a positive scalar in ohms.

Data Types: `double`

Z0o — Odd mode impedance

51 (default) | positive scalar

Odd mode impedance, specified as a positive scalar in ohms.

Data Types: `double`

Output Arguments

Length — Length of coupled-line section

positive scalar

Length of the coupled-line section, returned as a positive scalar.

Width — Width of coupled line section

positive scalar

Width of the coupled-line section, returned as a positive scalar.

Spacing — Spacing between coupled-line sections

positive scalar

Spacing between the coupled-line sections, returned as a positive scalar.

Version History

Introduced in R2022a

See Also

`designOutputLine` | `designUncoupledLine`

designUncoupledLine

Calculate dimensions of uncoupled-line section for specified frequency

Syntax

```
[Length,Width] = designUncoupledLine(balunobj,frequency)
___ = designUncoupledLine( ___,Name=Value)
```

Description

[Length,Width] = designUncoupledLine(balunobj,frequency) calculates the dimensions of the uncoupled-line section of a coupled line balun around a specified frequency. In the balunCoupledLine object, the uncoupled line dimensions are incorporated into the UncoupledLineShape property.

Note designUncoupledLine is the second step in designing a coupled line balun. This function is preceded by designCoupledLine and succeeded by designOutputLine.

___ = designUncoupledLine(___,Name=Value) calculates the dimensions of the uncoupled-line section of a coupled-line balun with additional options specified using name-value arguments.

Note PCB components designed using the design function operate around the specified frequency with a 10-15% tolerance.

Examples

Design Coupled Line Balun at 4 GHz

Define the frequency at 4 GHz.

```
f = 4e9;
```

Create a coupled line balun object.

```
balun = balunCoupledLine
```

```
balun =
    balunCoupledLine with properties:
```

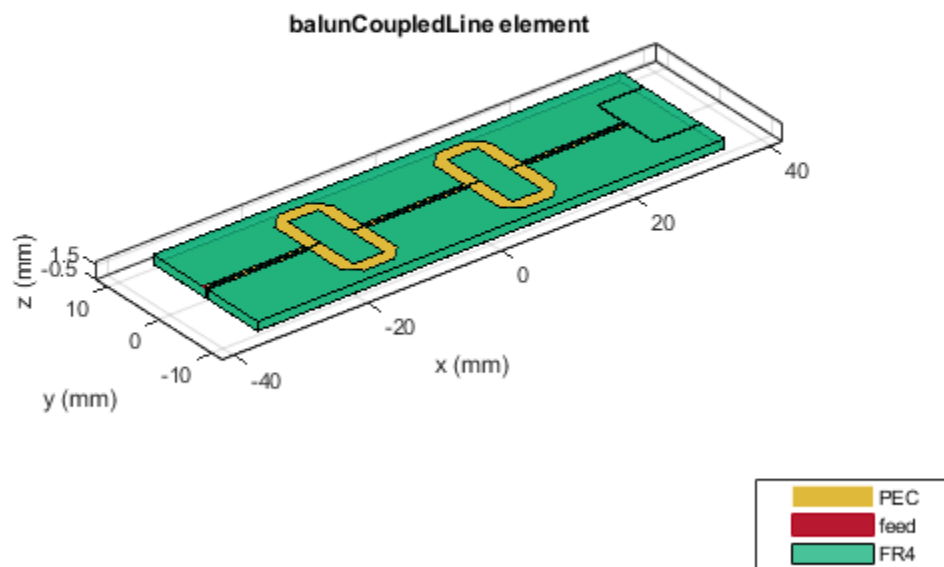
```
    NumCoupledLineSection: 3
    CoupledLineLength: 0.0153
    CoupledLineWidth: 4.0000e-04
    CoupledLineSpacing: 1.4000e-04
    UncoupledLineShape: [1x1 ubendMitered]
    OutputLineLength: 0.0124
    OutputLineWidth: 1.5300e-04
    OutputLineSpacing: 0.0110
```

```

Height: 0.0013
GroundPlaneWidth: 0.0200
Substrate: [1x1 dielectric]
Conductor: [1x1 metal]

```

```
show(balun)
```



Step 1: Design coupled line section

Design the coupled line section of the balun with an even mode impedance of 159 ohms and an odd mode impedance of 51 ohms. Use the helper function **designCoupledLine**.

```
[ClineL,ClineW,ClineS] = designCoupledLine(balun,f,'Z0e',159,'Z0o',51)
```

```
ClineL = 0.0107
```

```
ClineW = 4.2682e-04
```

```
ClineS = 1.4374e-04
```

Step 2: Design uncoupled line section

Design the uncoupled line section of the balun with the even and odd mode impedance of 59 ohms. Use the helper function **designUncoupledLine**.

```
[unclineL,unclineW] = designUncoupledLine(balun,f,'Z0',59,'LineLength',0.25)
```

```
unclineL = 0.0103
```

```
unclineW = 0.0018
```

Step 3: Design output line section

Design the output line section of the balun at the same frequency to extend the port 2 and port3. Use the helper function **designOutputLine**.

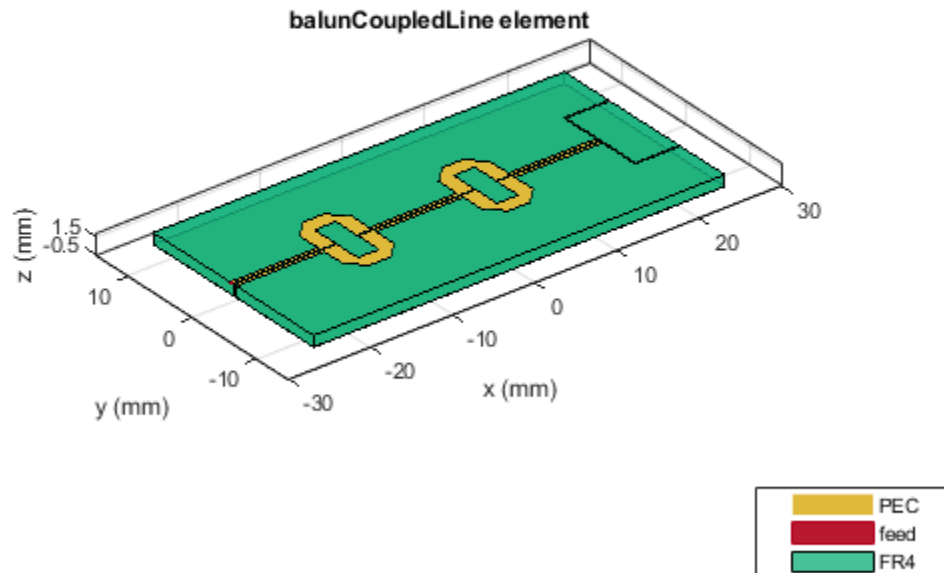
```
[OutL,OutW] = designOutputLine(balun,f,'Z0e',159,'Z0o',51,'Z0',59,'Zref',50)
```

```
OutL = 0.0109
```

```
OutW = 1.6115e-04
```

Set all the design dimensions to the coupled balun object.

```
balun.CoupledLineLength = ClineL;
balun.CoupledLineWidth = ClineW;
balun.CoupledLineSpacing = ClineS;
UncoupledLine = ubendMitered;
UncoupledLine.Length = [unclineL/2,unclineL/4,unclineL/2];
UncoupledLine.Width = [unclineW,unclineW,unclineW];
balun.UncoupledLineShape = UncoupledLine;
balun.OutputLineLength = OutL;
balun.OutputLineWidth = OutW;
balun.OutputLineSpacing = OutL+ClineS;
gndW = 25e-3;
balun.GroundPlaneWidth = gndW;
show(balun)
```

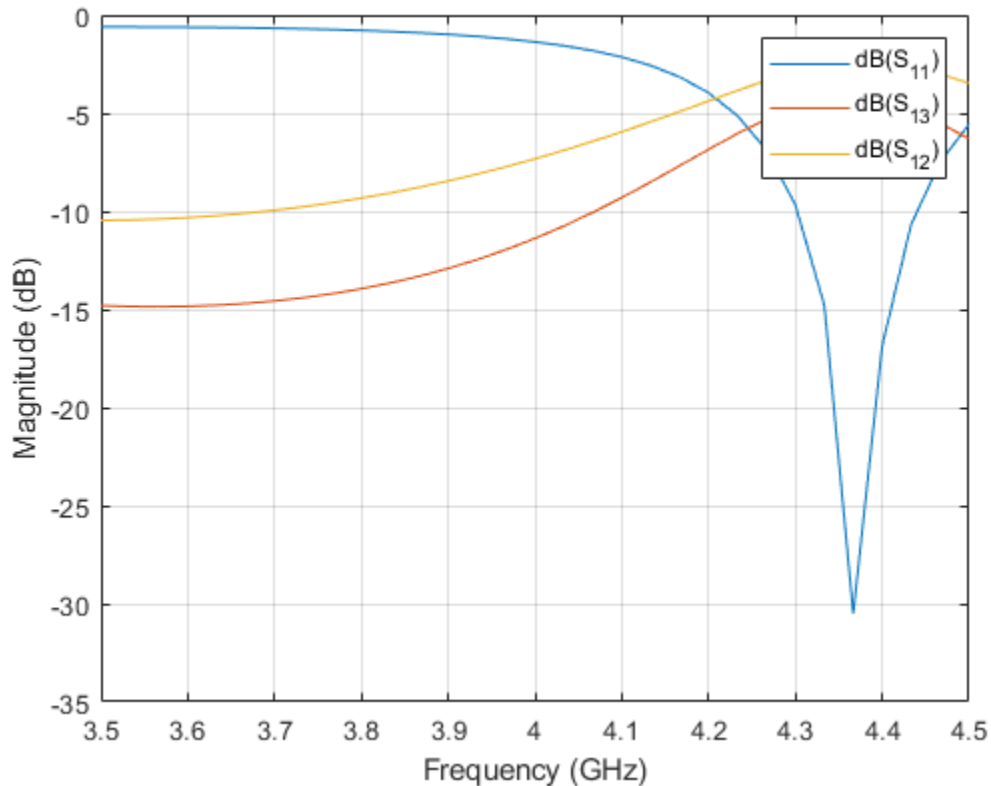


Analyze and plot the S-parameters of this balun.

```
s11 = sparameters(balun, linspace(3.5e9, 4.5e9, 31));
```

```
figure; rfplot(s11,1,1);
hold on; rfplot(s11,1,3)
hold on; rfplot(s11,1,2)
```

Click legend labels to toggle the line visibility



Input Arguments

balunobj — Coupled-line balun

balunCoupledLine object

Coupled-line balun, specified as a balunCoupledLine object.

Example: balunobj = balunCoupledLine; design(balunobj, 2e9) designs a coupled-line balun around a frequency of 2 GHz and returns the length and width of the uncoupled line section.

frequency — Design frequency of coupled-line balun

real positive scalar

Design frequency of coupled-line balun, specified as a real positive scalar in hertz.

Example: 55e6

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, . . . , NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=60`

Z0 — Impedance of uncoupled-line

50 (default) | positive scalar

Impedance of uncoupled-line, specified as a positive scalar in ohms.

Data Types: `double`

LineLength — Length of uncoupled-line

0.25 (default) | positive scalar

Length of uncoupled-line specified as a positive scalar in multiples of lambda. In the `balunCoupledLine` object, the uncoupled line dimensions are incorporated into the `UncoupledLineShape` property. `UncoupledLineShape.Length` is the vector of size 3 and the `LineLength` property should be splitted accordingly.

Data Types: `double`

Output Arguments

Length — Length of uncoupled-line section

positive scalar

Length of the uncoupled-line section, returned as a positive scalar.

Width — Width of uncoupled-line section

positive scalar

Width of the uncoupled-line section, returned as a positive scalar.

Version History

Introduced in R2022a

See Also

`designCoupledLine` | `designOutputLine`

designOutputLine

Calculate dimensions of output line section for specified frequency

Syntax

```
[Length,Width] = designOutputLine(balunobj,frequency)
___ = designOutputLine( ____,Name=Value)
```

Description

[Length,Width] = designOutputLine(balunobj,frequency) calculates the dimensions of the output line section of a coupled-line balun around a specified frequency.

Note designOutputLine is the third step in designing a coupled line balun. This function is preceded by designCoupledLine and designUncoupledLine as the first and second step, respectively.

___ = designOutputLine(____,Name=Value) calculates the dimensions of the output line section of a coupled-line balun with additional options specified using name-value arguments.

Note PCB components designed using the design function operate around the specified frequency with a 10-15% tolerance.

Examples

Design Coupled Line Balun at 4 GHz

Define the frequency at 4 GHz.

```
f = 4e9;
```

Create a coupled line balun object.

```
balun = balunCoupledLine
```

```
balun =
    balunCoupledLine with properties:
```

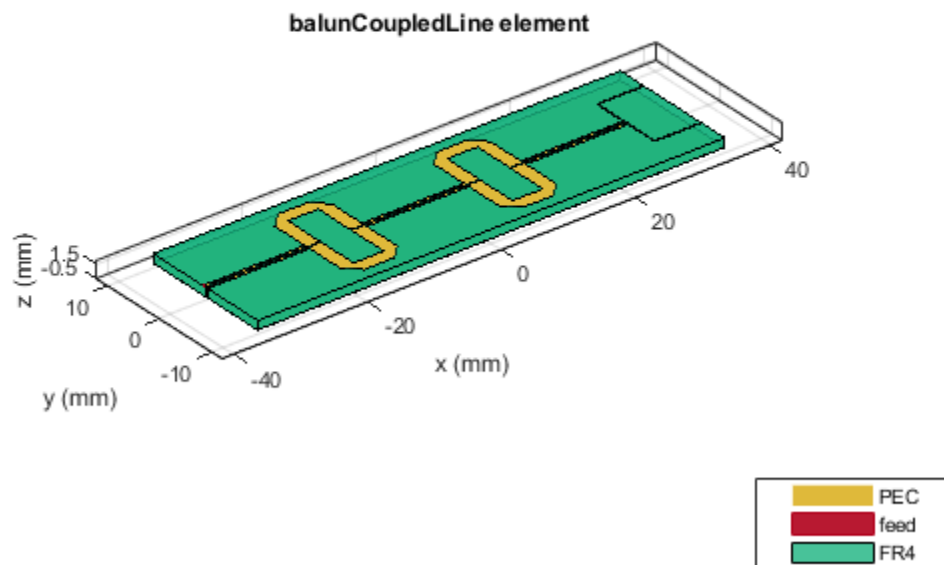
```
    NumCoupledLineSection: 3
    CoupledLineLength: 0.0153
    CoupledLineWidth: 4.0000e-04
    CoupledLineSpacing: 1.4000e-04
    UncoupledLineShape: [1x1 ubendMitered]
    OutputLineLength: 0.0124
    OutputLineWidth: 1.5300e-04
    OutputLineSpacing: 0.0110
    Height: 0.0013
```

```

GroundPlaneWidth: 0.0200
Substrate: [1x1 dielectric]
Conductor: [1x1 metal]

```

```
show(balun)
```



Step 1: Design coupled line section

Design the coupled line section of the balun with an even mode impedance of 159 ohms and an odd mode impedance of 51 ohms. Use the helper function **designCoupledLine**.

```
[ClineL,ClineW,ClineS] = designCoupledLine(balun,f,'Z0e',159,'Z0o',51)
```

```
ClineL = 0.0107
```

```
ClineW = 4.2682e-04
```

```
ClineS = 1.4374e-04
```

Step 2: Design uncoupled line section

Design the uncoupled line section of the balun with the even and odd mode impedance of 59 ohms. Use the helper function **designUncoupledLine**.

```
[unclineL,unclineW] = designUncoupledLine(balun,f,'Z0',59,'LineLength',0.25)
```

```
unclineL = 0.0103
```

```
unclineW = 0.0018
```

Step 3: Design output line section

Design the output line section of the balun at the same frequency to extend the port 2 and port3. Use the helper function **designOutputLine**.

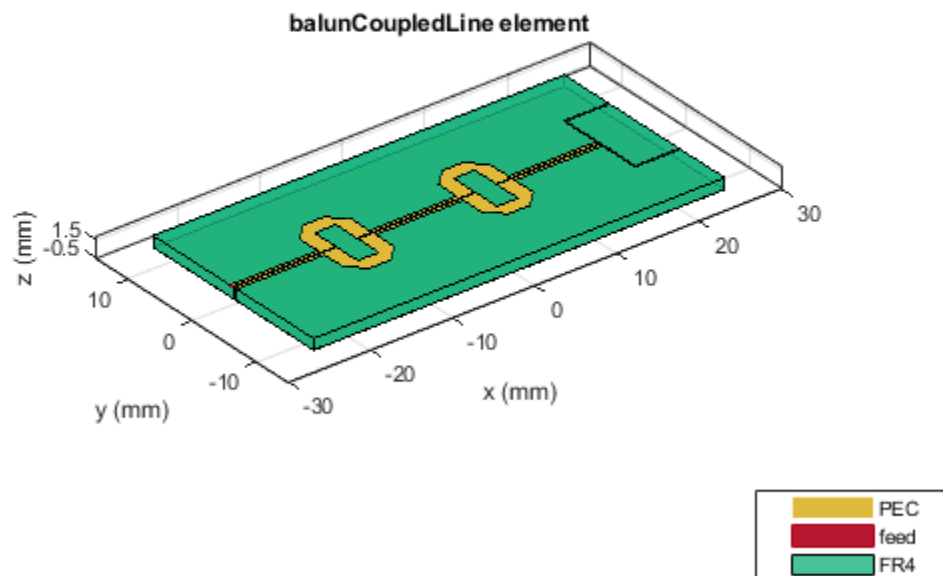
```
[OutL,OutW] = designOutputLine(balun,f,'Z0e',159,'Z0o',51,'Z0',59,'Zref',50)
```

```
OutL = 0.0109
```

```
OutW = 1.6115e-04
```

Set all the design dimensions to the coupled balun object.

```
balun.CoupledLineLength = ClineL;
balun.CoupledLineWidth = ClineW;
balun.CoupledLineSpacing = ClineS;
UnCoupledLine = ubendMitered;
UnCoupledLine.Length = [unclineL/2,unclineL/4,unclineL/2];
UnCoupledLine.Width = [unclineW,unclineW,unclineW];
balun.UnCoupledLineShape = UnCoupledLine;
balun.OutputLineLength = OutL;
balun.OutputLineWidth = OutW;
balun.OutputLineSpacing = OutL+ClineS;
gndW = 25e-3;
balun.GroundPlaneWidth = gndW;
show(balun)
```

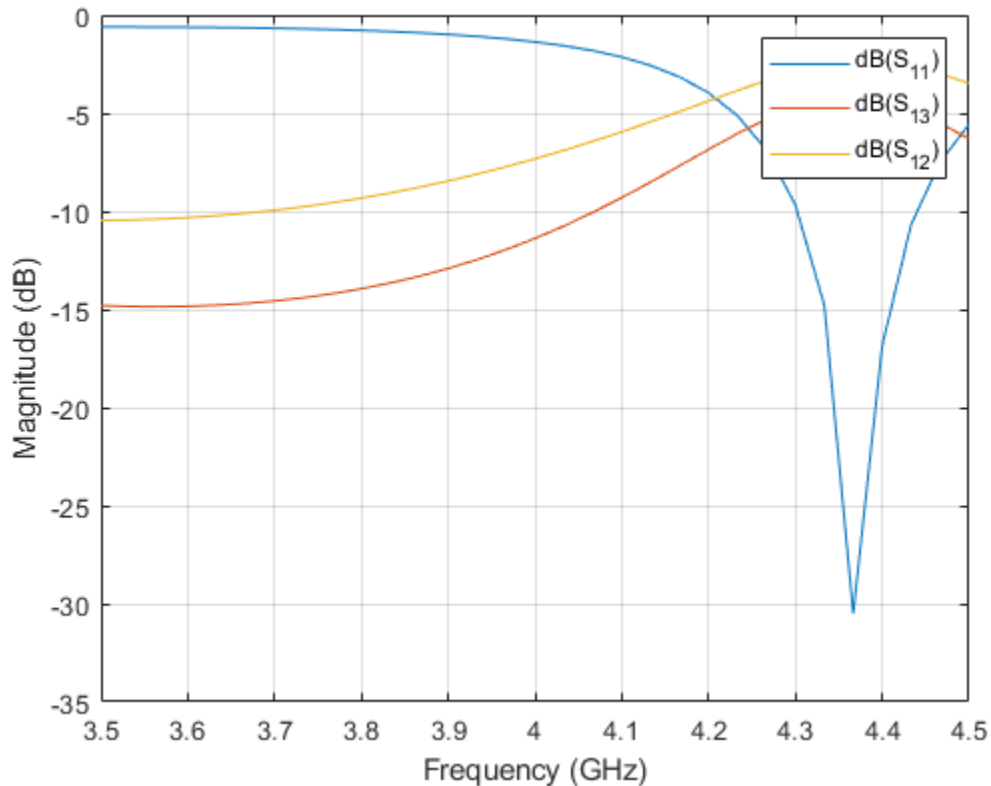


Analyze and plot the S-parameters of this balun.

```
s11 = sparameters(balun,linspace(3.5e9,4.5e9,31));
```

```
figure; rfplot(s11,1,1);
hold on; rfplot(s11,1,3)
hold on; rfplot(s11,1,2)
```

Click legend labels to toggle the line visibility



Input Arguments

balunobj — Coupled-line balun

balunCoupledLine object

Coupled-line balun, specified as a balunCoupledLine object.

Example: `balunobj = balunCoupledLine; design(balunobj,2e9)` designs a coupled-line balun around a frequency of 2 GHz and returns the `OutputLineLength` and `OuputLineWidth` properties of the coupled line balun.

frequency — Design frequency of coupled-line balun

real positive scalar

Design frequency of coupled-line balun, specified as a real positive scalar in hertz.

Example: `55e6`

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, . . . , NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0o=61`

Z0e — Even mode impedance of coupled-line

159 (default) | positive scalar

Even mode impedance of the coupled-line, specified as a positive scalar in ohms.

Note The even mode impedance should be same as the even mode impedance of the coupled line section.

Data Types: double

Z0o — Odd mode impedance of coupled-line

51 (default) | positive scalar

Odd mode impedance of the coupled-line, specified as a positive scalar in ohms.

Note The odd mode impedance should be same as the odd mode impedance of the coupled line section.

Data Types: double

Z0 — Impedance of uncoupled-line

50 (default) | positive scalar

Impedance of the uncoupled-line, specified as a positive scalar in ohms.

Note The impedance should be same as the impedance of the uncoupled line section.

Data Types: double

Zref — Reference impedance to match output

50 (default) | positive scalar

Reference impedance to match the output, specified as a positive scalar in ohms.

Data Types: double

Output Arguments

Length — Length of output line section

positive scalar

Length of the output line section, returned as a positive scalar.

Width — Width of output line section

positive scalar

Width of the output line section, returned as a positive scalar.

Version History

Introduced in R2022a

See Also

`designCoupledLine` | `designUncoupledLine`

design

Design wideband Wilkinson splitter around specified frequency

Syntax

```
wsplitter = design(wsplitterobj,frequency)
wsplitter = design( ____,Name=Value)
```

Description

`wsplitter = design(wsplitterobj,frequency)` designs a wideband Wilkinson splitter around the specified frequency.

`wsplitter = design(____,Name=Value)` designs a wideband Wilkinson splitter with additional options specified using name-value arguments.

Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

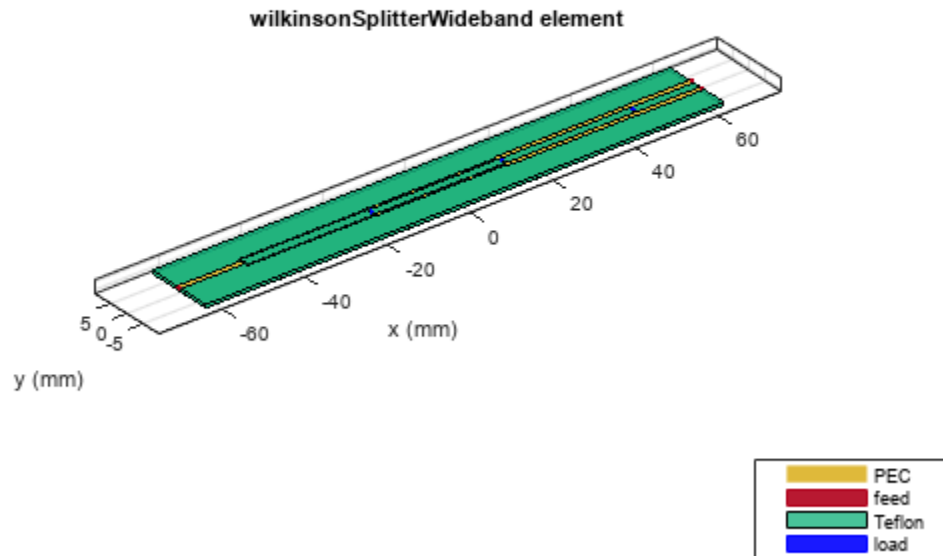
Design Wideband Wilkinson Splitter at 1.8 GHz

Design a wideband Wilkinson splitter at 1.8 GHz and with a Z0 of 75 ohm.

```
wsplitter = design(wilkinsonSplitterWideband,1.8e9,Z0=75);
```

View the splitter.

```
figure;
show(wsplitter);
```



Input Arguments

wsplitterobj — Wideband Wilkinson splitter

wilkinsonSplitterWideband object

Wideband Wilkinson splitter, specified as a wilkinsonSplitterWideband object.

Example: `wsplitterobj = wilkinsonSplitterWideband; design(wsplitterobj,2e9)` designs a wideband Wilkinson splitter around a frequency of 2 GHz.

frequency — Design frequency of wideband Wilkinson splitter

real positive scalar

Design frequency of the wideband Wilkinson splitter, specified as a real positive scalar in hertz.

Example: `55e6`

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=60`

Z0 — Characteristic impedance of splitter

50 (default) | positive scalar

Characteristic impedance of the splitter, specified as a positive scalar in ohms.

Data Types: double

Output Arguments**wsplitter — Wideband Wilkinson splitter operating around specified frequency**

wilkinsonSplitterWideband object

Wideband Wilkinson splitter operating around the specified frequency, returned as a wilkinsonSplitterWideband object.

Version History

Introduced in R2022a

See Also

sparameters

rlgc

Compute resistances, inductances, conductances, and capacitances

Syntax

```
rlgcparams = rlgc(txline, frequency)
```

Description

`rlgcparams = rlgc(txline, frequency)` returns the resistances, inductances (L), conductances (G), and capacitances per unit length of a transmission line at the specified frequency.

Examples

RLGC of Microstrip Transmission Line

Create a microstrip transmission line using a copper conductor.

```
txline = microstripLine;  
txline.Conductor.Name = 'Copper';  
txline.Conductor.Conductivity = 5.8e7;  
txline.Conductor.Thickness = 0.001;
```

Calculate the RLGC values of microstrip transmission line at 1 GHz.

```
freq = 1e9;  
RLGCparams = rlgc(txline, freq)
```

```
RLGCparams = struct with fields:  
  R: 1.4649  
  L: 2.0565e-07  
  G: 9.6413e-05  
  C: 9.5171e-11
```

Input Arguments

txline — Transmission line

transmission line object

Transmission line, specified as a `coupledMicrostripLine`, `microstripLine`, or `microstripLineCustom` object.

Example: `txline = microstripLine; rlgc(txline)` calculates the RLGC values of the microstrip transmission line object with handle `txline`.

Data Types: `char` | `string`

frequency — Frequency to calculate RLGC values

scalar

Frequency to calculate the RLGC values, specified as an integer in Hz.

Output Arguments

rlgcparams — RLGC values of transmission line

structure

RLGC values line per unit length of the transmission line, returned as a structure with field names for the R, L, G, and C values. Each field contains an n -by- n double precision matrix where n is the total number of traces in the transmission line object. Diagonal and nondiagonal entries in the matrix are self and coupled elements in the traces, respectively.

Version History

Introduced in R2022b

See Also

[coupling](#) | [directivity](#)

getZEven

Calculate even mode impedance of differential PCB transmission line

Syntax

```
zeven = getZEven(txline,frequency)
```

Description

`zeven = getZEven(txline,frequency)` calculates the even mode impedance of the transmission line at the specified frequency.

Examples

Even Mode Impedance of Coupled Microstrip Line

Create a coupled microstrip line with copper traces.

```
txline = coupledMicrostripLine;  
txline.Conductor.Name = 'Copper';  
txline.Conductor.Conductivity = 5.8e7;  
txline.Conductor.Thickness = 0.001;
```

Calculate the even mode impedance of coupled microstrip line at 1 GHz.

```
freq = 1e9;  
evenImp = getZEven(txline,freq)
```

```
evenImp = 49.1537
```

Input Arguments

txline — Transmission line

transmission line object

Transmission line, specified as a `coupledMicrostripLine`, `microstripLine`, or `microstripLineCustom` object.

Example: `txline = microstripLine;getZEven(txline)`. Calculates the even mode impedance of the microstrip transmission line object with handle `txline`.

Data Types: `char` | `string`

frequency — Frequency

positive scalar

Frequency to calculate the even mode impedance, specified as a positive scalar in hertz.

Output Arguments

zeven — Even mode impedance of transmission line

complex scalar

Even mode impedance of the transmission line, returned as a complex scalar.

Data Types: double

Version History

Introduced in R2022b

See Also

sparameters | getZ0dd | getZ0

getZOdd

Calculate odd mode impedance of differential PCB transmission line

Syntax

```
zodd = getZOdd(txline,frequency)
```

Description

`zodd = getZOdd(txline,frequency)` calculates the odd mode impedance of the transmission line at the specified frequency.

Examples

Odd Mode Impedance of Coupled Microstrip Line

Create a coupled microstrip line with copper traces.

```
txline = coupledMicrostripLine;  
txline.Conductor.Name = 'Copper';  
txline.Conductor.Conductivity = 5.8e7;  
txline.Conductor.Thickness = 0.001;
```

Calculate the odd mode impedance of coupled microstrip line at 1 GHz.

```
freq = 1e9;  
oddImp = getZOdd(txline,freq)
```

```
oddImp = 42.5414
```

Input Arguments

txline — Transmission line

transmission line object

Transmission line, specified as a `coupledMicrostripLine`, `microstripLine`, or `microstripLineCustom` object.

Example: `txline = microstripLine;getZOdd(txline)`. Calculates the odd mode impedance of the microstrip transmission line object with handle `txline`.

Data Types: `char` | `string`

frequency — Frequency

positive scalar

Frequency to calculate the odd mode impedance, specified as a positive scalar in hertz.

Output Arguments

zodd — Odd mode impedance of transmission line

complex scalar

Odd mode impedance of the transmission line, returned as a complex scalar.

Data Types: double

Version History

Introduced in R2022b

See Also

sparameters | getZEven | getZ0

design

Design phase shifter around specified frequency

Syntax

```
ps = design(psobj,frequency)
ps = design( ____,Name=Value)
```

Description

`ps = design(psobj,frequency)` designs a phase shifter around the specified frequency.

`ps = design(____,Name=Value)` designs a phase shifter with additional properties specified using name-value arguments.

Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

Design Phase Shifter at Specified Frequency

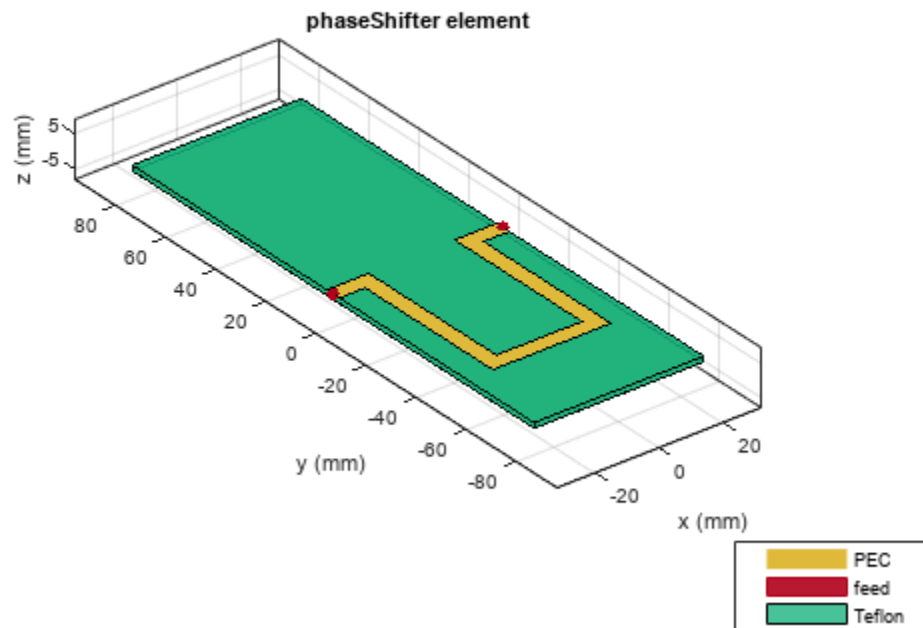
Design a phase shifter at 1.8 GHz with a phase shift of 75 degrees.

```
ps = design(phaseShifter,1.8e9,PhaseShift=75)
```

```
ps =
  phaseShifter with properties:
    NumSections: 1
    SectionShape: [1x1 ubendRightAngle]
    PortLineLength: 0.0080
    PortLineWidth: 0.0051
    Height: 0.0016
    GroundPlaneWidth: 0.1607
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

Show the phase shifter.

```
show(ps)
```

Input Arguments

psobj — Phase shifter
phase object

Phase shifter, specified as a `phaseShifter` object.

Example: `psobj = phaseShifter; design(psobj,2e9)` designs a phase shifter around a frequency of 2 GHz.

frequency — Design frequency of phase shifter
real positive scalar

Design frequency of the phase shifter, specified as a real positive scalar in hertz.

Example: `55e6`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=70`

Z0 — Characteristic impedance of phase shifter

50 (default) | positive scalar

Characteristic impedance of the phase shifter, specified as a positive scalar in ohms.

Data Types: double

PhaseShift — Phase shift value

90 (default) | positive scalar

Phase shift value, specified as a positive scalar in degrees.

Data Types: double

Output Arguments**psobj — Phase shifter operating around specified frequency**

phaseShifter object

Phase shifter operating around the specified frequency, returned as a phaseShifter object.

Version History**Introduced in R2022b****See Also**

sparameters

design

Design T-junction power splitter around specified frequency

Syntax

```
splitter = design(splitterteeobj,frequency)
splitter = design( ____,Name=Value)
```

Description

`splitter = design(splitterteeobj,frequency)` calculates the dimensions of a T-junction power splitter around the specified frequency.

`splitter = design(____,Name=Value)` designs a T-junction power splitter with additional options specified using name-value arguments.

Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

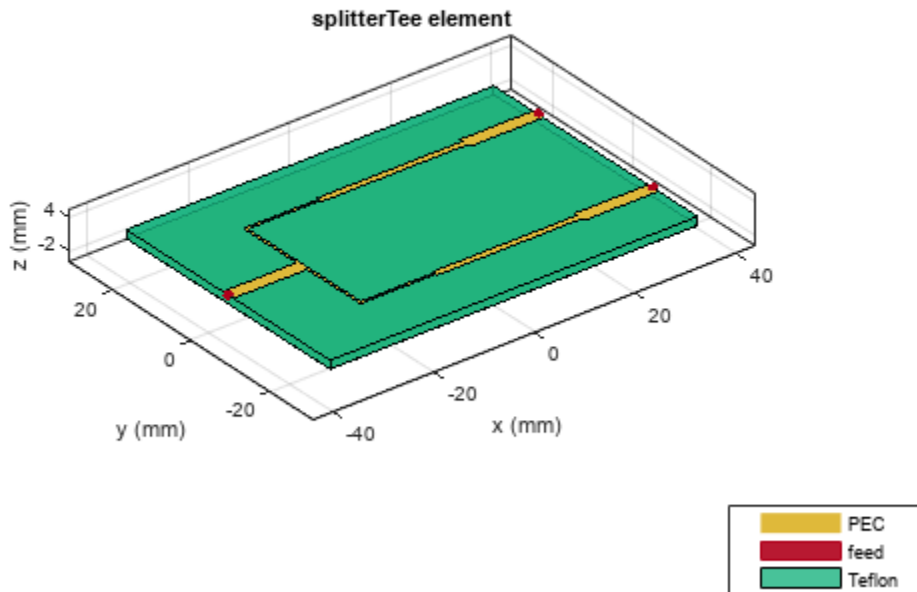
Design T-Junction Power Splitter

Design a T-junction power splitter at 2 GHz with a characteristic impedance Z_0 of 75 ohms.

```
splitter = design(splitterTee,2e9,Z0=75);
```

View the power divider.

```
show(splitter);
```



Input Arguments

splitterteeobj — T-junction power splitter

splitterTee object

T-junction power splitter, specified as a splitterTee object.

Example: `splitterteeobj = splitterTee; design(splitterteeobj,2e9)` designs a T-junction power splitter around a frequency of 2 GHz.

frequency — Design frequency of T-junction power splitter

real positive scalar

Design frequency of the T-junction power splitter, specified as a real positive scalar in hertz.

Example: `55e6`

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1, ..., NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `Z0=70`

Z0 — Characteristic impedance of T-junction power splitter

50 (default) | positive scalar

Characteristic impedance of the T-junction power splitter, specified as a positive scalar in ohms.

Data Types: double

Output Arguments**splitterteeobj — T-junction power splitter operating around specified frequency**

splitterTee object

T-junction power splitter operating around the specified frequency, returned as a splitterTee object.

Version History**Introduced in R2022b****See Also**

sparameters

design

Design ring resonator around specified frequency

Syntax

```
resonator = design(ringresonatorobj,frequency)
splitter = design( ____,Name=Value)
```

Description

`resonator = design(ringresonatorobj,frequency)` calculates the dimensions of a ring resonator around the specified frequency.

`splitter = design(____,Name=Value)` designs a ring resonator with additional options specified using name-value arguments.

Note PCB components designed using the `design` function operate around the specified frequency with a 10-15% tolerance.

Examples

Ring Resonator at 1.8 GHz

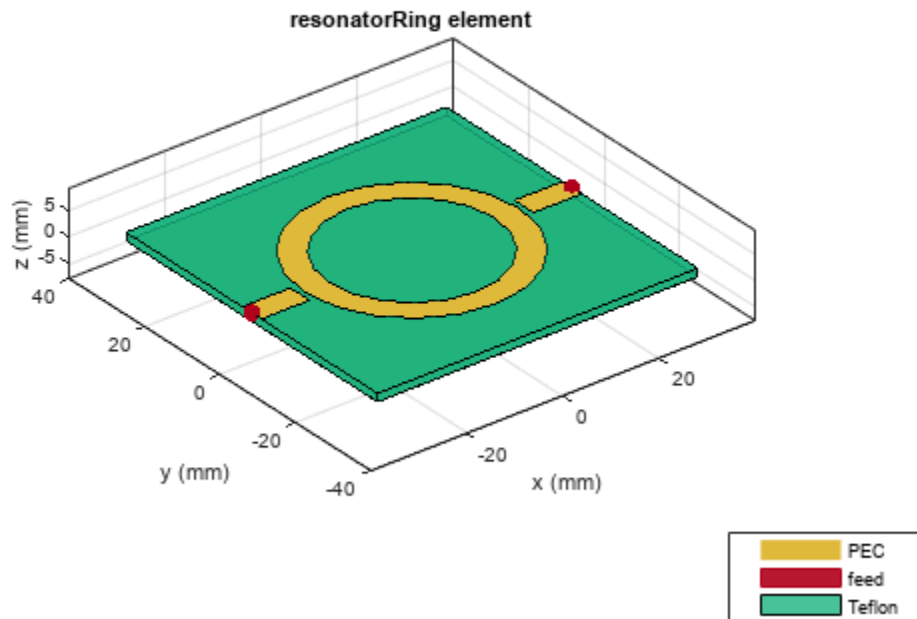
Design a ring resonator at 1.8 GHz.

```
resonator = design(resonatorRing, 1.8e9)
```

```
resonator =
    resonatorRing with properties:
        PortLineLength: 0.0100
        PortLineWidth: 0.0051
        CouplingGap: 1.0000e-03
        RingRadiusOuter: 0.0223
        RingWidth: 0.0051
        Height: 0.0016
        GroundPlaneWidth: 0.0669
        Substrate: [1x1 dielectric]
        Conductor: [1x1 metal]
```

View the resonator.

```
show(resonator);
```



Ring Resonator at 2.5 GHz

Design a ring resonator at 2.5 GHz with a characteristic impedance at 75 ohms.

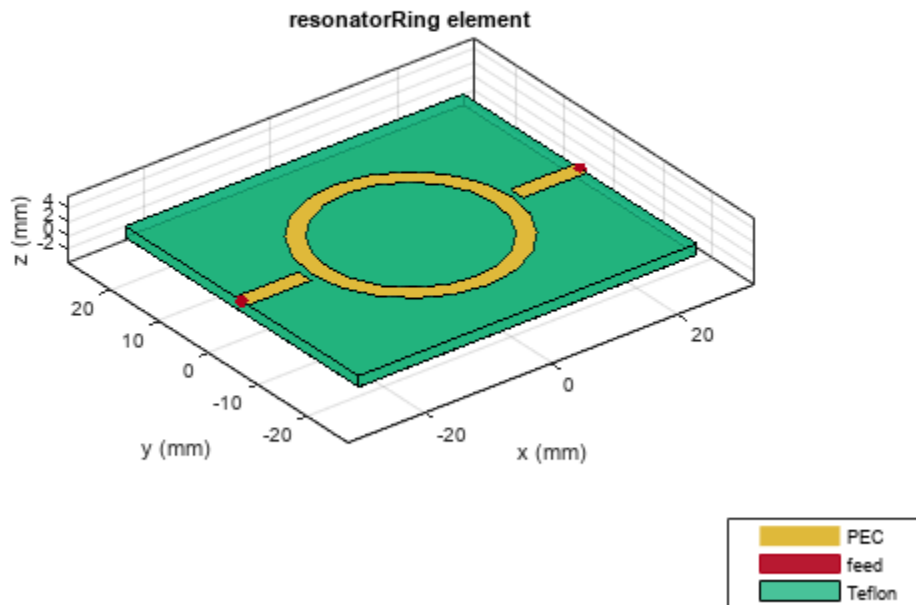
```
resonator = design(resonatorRing,2.5e9,Z0=75)
```

```
resonator =
  resonatorRing with properties:

    PortLineLength: 0.0100
    PortLineWidth: 0.0027
    CouplingGap: 1.0000e-03
    RingRadiusOuter: 0.0158
    RingWidth: 0.0027
    Height: 0.0016
    GroundPlaneWidth: 0.0474
    Substrate: [1x1 dielectric]
    Conductor: [1x1 metal]
```

View the resonator.

```
show(resonator)
```



Input Arguments

ringresonatorobj — Ring resonator
resonatorRing object

Ring resonator, specified as a resonatorRing object.

Example: resonator = resonatorRing; design(resonator, 2e9) designs a ring resonator around a frequency of 2 GHz.

frequency — Design frequency of ring resonator
real positive scalar

Design frequency of the ring resonator, specified as a real positive scalar in hertz.

Example: 55e6

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1, ..., NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: Z0=70

Z0 — Characteristic impedance of ring resonator

50 (default) | positive scalar

Characteristic impedance of the ring resonator, specified as a positive scalar in ohms.

Data Types: double

Output Arguments**resonator — Ring resonator operating around specified frequency**

resonatorRing object

Ring resonator operating around the specified frequency, returned as a resonatorRing object.

Version History

Introduced in R2022b

See Also

sparameters

propagationDelay

Compute propagation delay of transmission line

Syntax

```
pd = propagationDelay(txline, frequency)
```

Description

`pd = propagationDelay(txline, frequency)` returns the propagation delay of a transmission line

Examples

Propagation Delay of Microstrip Transmission Line

Create a microstrip transmission line with a copper trace.

```
m = microstripline;  
m.Conductor.Name = 'Copper';  
m.Conductor.Conductivity = 5.8e7;  
m.Conductor.Thickness = 0.001;
```

Calculate the propagation delay at 1 GHz.

```
freq = 1e9;  
pd = propagationDelay(m, freq)
```

```
pd = 4.4240e-09
```

Input Arguments

txline — Transmission line

transmission line object

Transmission line, specified as a `coupledMicrostripline`, `microstripline`, or `microstriplineCustom` object.

Example: `txline = microstripline; propagationDelay(txline)` calculates the propagation delay of the microstrip transmission line object with handle `txline`.

Data Types: `char` | `string`

frequency — Frequency to calculate propagation delay

scalar

Frequency to calculate the propagation delay, specified as an integer in Hz.

Output Arguments

pd — Propagation delay of transmission line

scalar double

Propagation delay of the transmission line in seconds per meter, returned as a scalar double.

Data Types: double

Version History

Introduced in R2022b

See Also

[coupling](#) | [directivity](#)

