100 Elwood Davis Road ♦ North Syracuse, NY 13212 ♦ USA

# Building a Sonnet Project Representing a Single Stub Circuit Using SonnetLab






**Specialists in High-Frequency Electromagnetic Software**
(315) 453-3096 Fax: (315) 451-1694 http://www.sonnetsoftware.com

Build Single Stub Circuit Using SonnetLab

In this tutorial we will build a single stub Sonnet project and simulate it.

The .m file for this tutorial can be found in

```
< SonnetLab directory>\Tutorials\Single Stub\Single Stub.m
```

The first thing we do in this tutorial is create a new Sonnet project. The command to make a new Sonnet project is SonnetProject(). The following command will make a new Sonnet project object and store it within the variable 'Project'.

```
Project=SonnetProject();
```

Now that the new project has been created Matlab can interact with it through the variable 'Project'. The new Sonnet project has the same parameters as a new project created with Sonnet. When creating a new Sonnet project in the Sonnet editor or with SonnetLab some aspects of the project must be specified by the user before the project can be simulated. One aspect of the Sonnet project that must be set is the thicknesses for the dielectric layers. A new Sonnet project will have two dielectric layers but neither will have a value for thickness; these values must be specified by the user. The thickness of the two dielectric layers can be changed with the following lines:

```
Project.changeDielectricLayerThickness(1,50);
Project.changeDielectricLayerThickness(2,5);
```

The first command changes the thickness of the first dielectric layer to a value of 50. The second command changes the thickness of the second dielectric layer to a value of 5. The units are specified globablly for a project (default is mils).

Now that the Sonnet project has valid dielectric layer properties our next step is to add the desired polygons to our project. The easiest way to add metal polygons to a Sonnet project is with the addMetalPolygonEasy() method. The addMetalPolygonEasy() method requires three arguments and may optionally take a fourth argument:

1. Metalization level index
2. The vector of X coordinate values
3. The vector of Y coordinate values
4. (Optional) The type of metal

In this tutorial we will not use the optional argument for the addMetalPolygonEasy() method. This argument is explained and used in the second single-stub tutorial. Because we did not specify this term the polygon is made out of lossless metal.

Sonnet projects with two dielectric layers have one metalization level. The first metalization level in a project is level zero. In this example there is only one metalization level so we will place our new polygons on level zero. The addMetalPolygonEasy() method also requires two vectors that specify the coordinates for the polygon verticies. The coordinates may be specified in either the clockwise or counterclockwise direction. The Sonnet project file format specifies that the last edge in a polygon must return to the first vertex of the polygon in order to close the shape. When using the addMetalPolygonEasy() method if the user doesn't close the polygon the method will do so automatically.

The layout this tutorial will implement will be to place a throughline along the width of the box and have a stub extend from the middle of the throughline towards the top of the box. A via to ground will be attached to the stub. The vertex coordinate vectors for the throughline are the following:

```
anArrayOfXCoordinates=[0;160;160;0];
anArrayOfYCoordinates=[130;130;150;150];
```

The throughline can be added to level zero of the the project with the following call to the addMetalPolygonEasy() method:

```
Project.addMetalPolygonEasy(0,anArrayOfXCoordinates,anArrayOfYCoordinates);
```

The same sequence of commands may be used to create the stub:

```
anArrayOfXCoordinates=[70;90;90;70];
anArrayOfYCoordinates=[130;130;20;20];
Project.addMetalPolygonEasy(0,anArrayOfXCoordinates,anArrayOfYCoordinates);
```

In this design we will add a square via polygon to the end of our stub; this can be accomplished using the addViaPolygonEasy function. The addViaPolygonEasy() method requires four values and may optionally take a fifth argument:

1. Metalization level index
2. The level the via is attached to
3. The vector of X coordinate values
4. The vector of Y coordinate values
5. (Optional) The type of metal

In this tutorial we will not use the optional argument for the addViaPolygonEasy() method. This argument is explained and used in the second single-stub tutorial. Because we did not specify this term the polygon will be made out of lossless metal.

In this example we want our via polygon to run from metalization level zero to the ground plane so we will specify the value for the second argument to be 'GND'. The second argument could also be a number to indicate another level. For example if a user wanted to connect a via between level zero and one then the second function argument should be the number one. The commands to add the via polygon to the project are displayed below:

```
anArrayOfXCoordinates=[70;90;90;70];
anArrayOfYCoordinates=[20;20;40;40];
Project.addViaPolygonEasy(0,'GND',anArrayOfXCoordinates,anArrayOfYCoordinat
es);
```

With our polygons in place we may proceed to adding ports to the throughline. SonnetLab has serveral methods for adding ports to projects; one of the easiest approaches is the addPortAtLocation() method.

The addPortAtLocation() function takes an (X,Y) coordinate pair as arguments and attempts to place a port at the nearest polygon edge. In this tutorial we want to add a port to the throughline at location (0,140) and another port at location (160,140). These two ports can be added with the following lines:

```
Project.addPortAtLocation(0,140);
Project.addPortAtLocation(160,140);
```

Congradulations! The layout for the single stub circuit is complete. Although the components of the circuit have been created there are a few more steps that need to be taken before we can simulate the project.

Before the project can be simulated the user must first specify the frequency sweep settings. In this tutorial we will simulate the project using Sonnet's ABS simulation routine. The two arguments that the ABS routine needs is a start frequency value and a stop frequency value. The command to add an ABS frequency sweep from 5Ghz to 10Ghz is the following:

```
Project.addAbsFrequencySweep(5,10);
```

When a new project is created using SonnetLab it will need to be saved to the hard drive before it can be simulated. The first time a user wants to save a particular Sonnet project to the hard drive they must specify a filename for the project; this can be accomplished using the saveAs() method as follows:

```
Project.saveAs('SingleStub.son');
```

The above command will save the Sonnet project specified by the variable Project to the hard drive as 'SingleStub.son'. After the user does a single saveAs() command they are then able to use the save() command which will save the file to the hard drive using the same filename that was specified by the most recent call to saveAs(). If a Sonnet project file is opened from the hard drive (rather than being created from scratch from within Matlab) the user does not need to call saveAs() in order to specify a filename for the project file; the save() function will overwrite the read project file. At any time any Sonnet project may be saved with saveAs() in order to specify a different filename; the result of which will also cause all later calls of save() to be saved to the new filename.

Now that we have specified the simulation settings we can call Sonnet's simulation engine to simulate the project using the command:

```
Project.simulate();
```

When the simulation is complete the user may examine the results using the Sonnet response viewer. SonnetLab can automate opening the Sonnet response viewer with the following command:

```
Project.viewResponseData();
```