SonnetLab DStub Tutorial

By Bashir Souid

This work was supervised by Dr. Serhend Arvas.

This tutorial will mimic the design approach used in chapter 5 of the Sonnet "Getting Started Guide". The guide demonstrates a lot of the core functionalities of the Sonnet circuit designer. This tutorial follows all the steps completed in the "Getting Started Guide". There are points where Sonnet's guide will make a change to a project and then change it back to its original setting; this tutorial will also make both changes in order to show the user how to complete those operations in SonnetLab.

The .m file for this tutorial can be found in

```
<SonnetLab directory>\Tutorials\DStub\DStub.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 the Sonnet GUI. New Sonnet Projects have a box size of 160 by 160 mils. In this tutorial the first thing we want to do is change the box size for our project to be 330 by 200 mils. This change can be made with the following command:

```
Project.changeBoxSize(330,200);
```

The default cell size for a new Sonnet project is 10 by 10 mils. We want our design to have a cell size of 10 by 10 mils so no change to the project is necessary. Although the cell size does not need to be set this tutorial will present the way to change the cell size. The following line will explicitely set the cell size of the project to 10 by 10 mils by modifying the number of cells:

```
Project.changeCellSizeUsingNumberOfCells(10,10);
```

When the cell size is changed either the size of the box or the number of cells in the box must change in order to realize the new cell size. In most cases it is desireable change the cell size by changing the number of cells that span the length and width of the box.

One of the first things a designer will typically modify for a design is the settings for the dielectric layers. A new Sonnet project will have two dielectric layers but neither will have a value for thickness; the thickness must be specified by the user. The thickness of the first dielectric layer can be changed with the following line:

```
Project.changeDielectricLayerThickness(1,20);
```

The above command changes the thickness of the first dielectric layer to a value of 20. The units are specified globablly for a project (default is mils).

We would like the second layer of our project to have the following settings:

```
Name: Alumina
Thickness: 20
Erel: 9.8
Dielectric Loss Tangent: 1.0e-4
Dielectric Conductivity: 0.0
Mrel: 1.0
Magnetic Loss Tangent: 0.0
```

The easiest way to change the second level of our project to be the above values is to delete the second layer that was automatically generated for our project and add a new level with the desired values to the project. The second level can be deleted with the following command:

```
Project.deleteLayer(2);
```

A layer with the desired properties can then be added to the project using the following command:

```
Project.addDielectricLayer('Alumina',20,9.8,1,1.0e-4,0,0);
```

The next thing we want to do in this tutorial is to add metal polygons to our project to represent a DStub. In order to build a DStub circuit we will first need a throughline to go from the left side of the box to the right side. The easiest way to add metal polygons to a Sonnet project is with the addMetalPolygonEasy() method. The addMetalPolygonEasy() method requires three values 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 Metal type

The fourth argument for the addMetalPolygonEasy() method specifies the type of metal that should be used to create the metal polygon. We will omit the fourth argument for this tutorial; when the fourth argument is omitted the polygon is made out of lossless metal. The addMetalPolygonEasy() method will return a reference to the newly added polygon.

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 which is level zero. The addMetalPolygonEasy method also requires two vectors to 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 point in a polygon must return to the first vertex of the polygon in order to close the shape. If the user doesn't close off the polygon then addMetalPolygonEasy will do so automatically. The vertex coordinate vectors are assigned with the following two lines:

```
aVectorOfXCoordinates=[0 330 330 0];
aVectorOfYCoordinates=[90 90 110 110];
```

And the polygon is added to the project with the following line.

```
aThroughLine=Project.addMetalPolygonEasy(0,aVectorOfXCoordinates,aVectorOfY
Coordinates);
```

The above command added a throughline to our design. The same sequence of commands may be

used to create a stub:

```
aVectorOfXCoordinates=[60 60 80 270 270 80 80 60];
aVectorOfYCoordinates=[110 130 150 150 130 130 110 110];
aBottomStub=Project.addMetalPolygonEasy(0,aVectorOfXCoordinates,aVectorOfYC
oordinates);
```

The Sonnet "Getting Started Guide" then proceeds to copy the metal polygon, flip it, and move it to the other side of the throughline. We could have added the second stub to the project using the same method as we used to make the first stub but this tutorial will instead make a copy of the stub and flip/move it appropriately. The stub can be copied using the following command:

```
aTopStub=Project.duplicatePolygon(aBottomStub);
```

The above command will make a copy of the specified polygon, add it to the project, and return a reference to the newly created polygon.

The next thing we want to do is flip the newly copied polygon. We can flip the polygon in both the X and Y directions by using the following commands:

```
aTopStub.flipPolygonX();
aTopStub.flipPolygonY();
```

Now the only thing left to do to the stub is move it to be adjacent to the throughline. This move can be accomplished by the following command:

```
aTopStub.movePolygon(165,70);
```

The movePolygon() method will move the polygon such that its centroid is at the location (165,70). The polygon's centorid is the middle of the range of X coordinates and the middle of the range of Y coordinates.

All of the polygons in our project at this point have been lossless metal. One of the things the Sonnet "Getting Started Guide" shows teaches users is how to define a new type of metal, change the type of metal polygons in a project to the new metal type and how to change them back to lossless. This tutorial will also perform those operations even though the net result of those operations is an identical design.

Before we can use a user defined metal type we must first define it. In this tutorial we want to define a metal type called 'Half Oz Copper' that has the following properties:

```
Name: Half Oz Copper
Type: Normal
Conductivity: 5.8e7
Thickness: .7
```

This half oz copper metal type can be defined with the following command:

```
Project.defineNewNormalMetalType('Half Oz Copper',5.8e7,0,.7);
```

Now that our new metal type is defined we can proceed to changing the three polygons in our project to this new metal type. There are different ways to change the type for a metal polygon in a project but in this tutorial we will use the `changePolygonType()` method which will accept a polygon reference. We can change all three metal polygons in our project to half oz copper with the following commands:

```
Project.changePolygonType(aThroughLine,'Half Oz Copper');
Project.changePolygonType(aTopStub,'Half Oz Copper');
Project.changePolygonType(aBottomStub,'Half Oz Copper');
```

We can then change the metal types of these polygons back to lossless using the following three commands:

```
Project.changePolygonType(aThroughLine,'Lossless');
Project.changePolygonType(aTopStub,'Lossless');
Project.changePolygonType(aBottomStub,'Lossless');
```

The lossless metal type is built into the Sonnet project format and does not need to be defined before it can be used.

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 methods is with the `addPortAtLocation` function.

The `addPortAtLocation` function takes two arguments: a X coordinate and a Y coordinate. The X and Y coordinate indicate the location where the user would like to place the port (must be near a polygon edge) and SonnetLab will place the port in the appropriate location. In this example we want to add a port to the throughline at location (0,100) and another port at location (330,100). These two ports can be added with the following lines:

```
Project.addPortAtLocation(0,100);
Project.addPortAtLocation(330,100);
```

Congradulations! We have successfully designed a DStub circuit. 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.

When a new project is created 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('DStub.son');
```

The above command will save the Sonnet project specified by the variable Project to the hard drive with a file named 'DStub.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.

Before the project can be simulated the user must first specify the frequency sweep settings. In this tutorial we will simulate the project using a linear frequency sweep. The three arguments that are needed for linear frequency sweeps are the start frequency value, a stop frequency value and the step size. The command to add a linear frequency sweep from 4Ghz to 8Ghz with a step size of .25Ghz is the following:

```
Project.addSimpleFrequencySweep(4.0,8.0,0.25);
```

Now that we have specified the simulation settings we can simulate the project using the command

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

The above command will save the project and call Sonnet's simulation engine to simulate the project.

When the simulation is complete the user may examine the results by manually opening the Sonnet response viewer or they can automate opening the response viewer by issueing the following command:

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