SonnetLab Double Stub Tutorial

By Bashir Souid

This work was supervised by Dr. Serhend Arvas.

This tutorial will cover most aspects of the single stub tutorial and incorporate additional methods and functionality. Although this tutorial is intended to be taken after the single stub tutorial this tutorial re-enforces many of the previous concepts.

The .m file for this tutorial can be found in

```
< SonnetLab directory>\Tutorials\Double Stub\DoubleStub.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 changed the thickness of the second dielectric layer to a value of 5. The units are specified globablly for a project (default is mils).

A default Sonnet project has a box of size 160x160. This value will need to be changed for most circuit designs. The user may change the size of the box easily using the following command:

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

The above command changed the size of the box in the X direction to 200 and the size of the box in the Y direction to be 200. This enlargement of the box does not change the number of cells in the X and Y directions; changing the size of the box changes the cell size rather than the number of cells. If we would like to have a cell size of 10x10 then we can use the following function to change the number of cells in the X and Y directions such that the cell size is the desired size.

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

In the single stub tutorial the polygons were made out of lossless material. In this tutorial we will be using copper polygons. Before using copper polygons in a Sonnet project the copper type of metal must be added to the project; this can be done with the following command:

```
Project.defineNewNormalMetalType('Copper',58000000,0,1.4);
```

The above command will add the copper metal type to the project with conductivity 58000000, current ratio 0 and thickness of 1.4.

Now that we have defined the copper metal type our next step is to add the desired copper 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 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

addMetalPolygonEasy() may accept a fourth parameter which specifies the type of metal that should be used to create the metal polygon. The type of the metal to be used must be a previously defined metal type. In the previous code snippit we defined a copper metal type for this project. The metal type argument may either be the index of the metal type in the array of metal types or the name for the metal type. It will be much easier for most users to simply use the name for the metal type.

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 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 he first vertex of the polygon in order to close the shape. If the user doesn't close the polygon then addMetalPolygonEasy will do so automatically. The vertex coordinate vectors are assigned with the following two lines:

```
anArrayOfXCoordinates=[0;200;200;0];
anArrayOfYCoordinates=[130;130;150;150];
```

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

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

The above command added the throughline for our design. The same sequence of commands may be used to create the stubs:

```
anArrayOfXCoordinates=[65;80;80;65];
anArrayOfYCoordinates=[130;130;20;20];
Project.addMetalPolygonEasy(0,anArrayOfXCoordinates,anArrayOfYCoordinates,'
Copper');

anArrayOfXCoordinates=[85;100;100;85];
anArrayOfYCoordinates=[130;130;20;20];
Project.addMetalPolygonEasy(0,anArrayOfXCoordinates,anArrayOfYCoordinates,'
Copper');
```

In this design we will add a square via polygon to the end of our stubs; this can be accomplished using the addViaPolygonEasy function. The addViaPolygonEasy function requires four arguments and may optionally accept a fith:

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 metal type

In this example we want our via polygons 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 a user wanted to connect a via between level zero and one they would specify the second argument to be the number one. We are going to make our via polygons out of copper rather than lossless metal. The commands to add the via polygons are displayed below:

```
anArrayOfXCoordinates=[65;80;80;65];
anArrayOfYCoordinates=[20;20;40;40];
Project.addViaPolygonEasy(0,'GND',anArrayOfXCoordinates,anArrayOfYCoordinat
es,'Copper');

anArrayOfXCoordinates=[85;100;100;85];
anArrayOfYCoordinates=[20;20;40;40];
Project.addViaPolygonEasy(0,'GND',anArrayOfXCoordinates,anArrayOfYCoordinat
es,'Copper');
```

The polygons for our stubs and vias are not on the grid-lines for the project. Sonnet's simulation engine does not simulate off-grid polygons. The Sonnet simulation engine will snap all polygons to the grid before it simulates a circuit. It is benifical to have your polygons on grid-lines to avoid the confusion that comes from having your circuit look one way and simulate in a different way. The following command will snap all polygons in the project to the grid:

```
Project.snapPolygonsToGrid();
```

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 two arguments: a X and 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 through line at location (0,140) and another port at location (200,140). These two ports can be added with the following lines:

```
Project.addPortAtLocation(0,140);
Project.addPortAtLocation(200,140);
```

Congradulations! The double stub circuit has now been created. It is easy to see our final design before simulating by entering the following command:

```
Project.drawCircuit();
```

The above command will draw a Matlab 3D plot of the circuit in a figure window.

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

example 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(5,50);
```

In this tutorial we will add an output file to our project using the addFileOutput method. Output files are a great way to export simulation data into a format that can be easily read by Matlab or other tools. In this tutorial we are going to add a touchstone file to the project using the following command:

```
Project.addTouchstoneOutput();
```

The above command added a touchstone file output of the S-parameter data to the project.

Before the circuit is simulated we will enable current calculations. Enabling current calculations will increase simulation time.

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('DoubleStub.son');
```

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

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

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

The circuit can be opened in Sonnet with the following command:

```
Project.openInSonnet(false);
```

The openInSonnet method will open the circuit in Sonnet's circuit tool. The argument given to openInSonnet is optional. If the argument is not passed or if the argument is boolean true then Matlab will save the project, open the project in the Sonnet GUI and wait for the Sonnet GUI to be closed. When the Sonnet GUI is closed Matlab will re-read the project file and update the Matlab version of the project to reflect any changes saved by Sonnet. If openInSonnet is passed a boolean false then Matlab will save the project, open the project in the Sonnet GUI, and continue execution.

When false is sent to openInSonnet the changes saved by Sonnet will not be updated in the version of the project that exists in Matlab.

SonnetLab can automate opening the Sonnet response viewer by executing the following command:

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

The current data can be viewed in Sonnet's current imagining tool by executing the following command:

```
Project.viewCurrents();
```