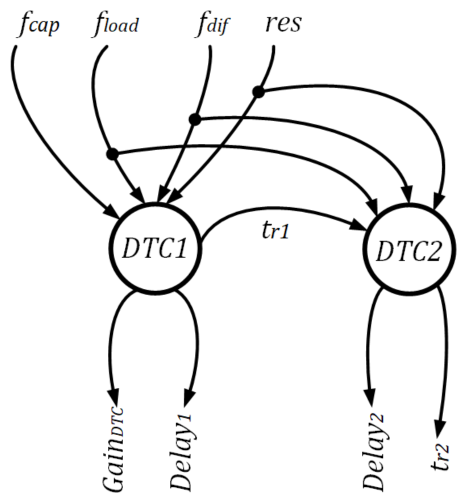
AMPSE Tutorial (top level gradient descent)

Qiaochu Zhang, Mike Chen's Mixed Signal Group, USC

1. Assuming you have already generated the regression models for all the modules in your block.
2. Update your workarea. You should have a new folder called GlobalLibrary inside workarea_POSH. Change the directory of the folder in the AMPSE_graph.py

```
sys.path.insert(0,'/your_directory/workarea_POSH/GlobalLibrary')
```

3. Draw the graph of your circuit block using the modules to show the relationship between design parameters, module level design metrics and top level design specifications. Here is one example of graph.



4. Define the classes of all the modules following the instruction in the script of AMPSE_graph.py
   - Modify the names of variables and directories that stores regression models
   - Modify the ranges of search parameters
5. Define the graph in script.
   - Examples
   1) The inputs of a module come from independent design parameters.

```
sx_DTC1 = sxin[:,0:4] # randomly generated parameters, scaled between -1 to 1
x_DTC1  = DTC1.tf_rescalex(sx_DTC1) # rescale input parameters back to real value
sy_DTC1 = DTC1.tf_reg_relu(sx_DTC1) # calculate scaled metric for module DTC1
y_DTC1  = DTC1.tf_rescaley(sy_DTC1) # rescale output metrics back to real value
```

   2) The inputs of a module depend both on design parameters and output of other modules.

```
cnst_DTC2 =
DTC2.tf_scalex(tf.concat([y_DTC1[:,2],x_DTC1[:,1],x_DTC1[:,2],x_DTC1[:,3]],axis=0))
# concatenate all the real values of the inputs for DTC2, and scale it with the standard scaler
of DTC2

sx_DTC2 = cnst_DTC2 # define the input of DTC2
sx_DTC2 = tf.reshape(sx_DTC2,(1,4))  # reshape the inputs of DTC2
x_DTC2 = DTC2.tf_rescalex(sx_DTC2) # rescale input parameters back to real value
sy_DTC2 = DTC2.tf_reg_relu(sx_DTC2) # calculate scaled metric for module DTC1
y_DTC2 = DTC2.tf_rescaley(sy_DTC2) # rescale output metrics back to real value
```

6. Define specifications and constraints

```
dtc_gain = y_DTC1[0,0] - y_DTC1[0,1]
dtc_offset = y_DTC1[0,1] + y_DTC2[0,0]
tr = y_DTC2[0,1]

specs = []
specs.append((dtc_gain-120e-12)) # 0- DTC gain
specs.append((dtc_offset-130e-12)) # 1- DTC offset
specs.append((tr-6e-11)) # 2- rise time

constraints = []
constraints.append(tf.nn.elu(specs[0]/DTC1.scYscale[0]))
constraints.append(tf.nn.elu(specs[0]*-1/DTC1.scYscale[0]))
constraints.append(tf.nn.elu(specs[1]/DTC1.scYscale[1]))
constraints.append(tf.nn.elu(specs[1]*-1/DTC1.scYscale[1]))
constraints.append(tf.nn.elu(specs[2]/DTC2.scYscale[1]))
constraints.append(tf.nn.elu(specs[2]*-1/DTC2.scYscale[1]))
```

7. Modify the following script with the instruction in the comments
8. After you run the simulation, all the candidates are stored accordingly. All the design parameters are stored at lst_params. All the metrics are stored at lst_metrics. All the specs are stored at lst_specs. You can postprocess them based on your requirement.
9. Run the script and have fun!