AMPSE File Hierarchy

NDA protected

Open Source

Main Folder

1- Netlists_Database.py
2- APMSE_Graphs.py
3- APMSE_CircTest.py

Desanitizer

netlists_desanitized

netlists_sanitized

datasets

reg_files

make_reg

AMPSE File Hierarchy

- Python Codes:
  - Netlist_Database.py:
    - Links python to all necessary SPICE testbenches
  - AMPSE_Regsearch.py:
    - Searching the parameters globally in regression models
  - AMPSE_TestCircuit.py:
    - AMPSE Verification on circuit
- Folders:
  - netlists (sanitized vs desantized)
  - datasets
  - reg_files
  - make_reg

Spectre Testbench (example)

```
0- // Generated by Cadence – spectre
…
4- // Design view name: schematic
5- simulator lang=spectre
6- global 0
7- parameters  rres=2K fnnn=15 fppp=5 VBIAS=0.9 wpppn=670n wnnn=390n lastt=60n
8- include "/technology65nm.scs" section=MC_TT
…
260- simulator lang=spice   // for better compatibility with Xyce
261- .MEAS TRAN_CONT cont_vout1 find v(vbias) when v(oo\<1\>)=0.8 rise=1
262- .MEAS TRAN_CONT cont_tout1 when v(oo\<1\>)=0.8 rise=1
```

SPICE Output (preprocessed results)

```
$DATA1 SOURCE='AFS'
VERSION='2017_Q1_update4'
Measurement results:1.
cont_tout1, result=  2.2153e-11
cont_tout1, result=  2.7812e-10
cont_tout1, result=  5.3852e-10
cont_tout1, result=  7.9873e-10
cont_tout1, result=  1.0588e-09
cont_tout1, result=  1.3186e-09
cont_tout1, result=  1.5784e-09
cont_tout1, result=  1.8382e-09
cont_tout1, result=  2.0982e-09
                    ¦
                    ¦
                    ¦
                    ¦
                    ¦
```

Calculate Metrics from Testbench in Repository

- Four primitive functions:
  1. Initialization:
     - Set parameters/metrics name,
     - Parameters range
     - Test-bench location
     - Work-area folder
  2. Run SPICE with given parameters
     - Runs Spectre/AFS/Xyce
     - Saves pre-processed data into list of arrays in Python
  3. Metrics calculation in Python
     - Calculate metrics from saved preprocessed data
  4. Run SPICE + Metrics calculation

Before making the Class import TestSpice and Netlists

```
from spectreIOlib import TestSpice, Netlists
```

Primitive function 1. Initialization

```
from spectreIOlib import TestSpice, Netlists

Class YOUR_MODULE_NAME(Netlists):
      def __init__(self, tech = 65, testfolder = None):
            if tech = 65:
                  self.testbench = YOUR_TESTBENCH
                  self.testfolder  = YOUR_TESTFOLDER
                  self.minpar = PARAMETER_MINVALUES # np.array 1D
                  self.maxpar = PARAMETER_MAXVALUES # np.array 1D
                  self.stppar = PARAMETER_STEPSIZES # np.array 1D
                  self.parname = PARAMETER_NAMES    # list
                  self.par_line_number = LINEINTESTBENCH # Integer
                  self.lst_metrics = …              # List of Dictionary
                  self.runspectre = TestSpice(…)    # TestSpice Instance
                  self.finaldataset = YOURDATASET.csv
```

Primitive function 1. Initialization (Example)

```
class VCOSpice(Netlists):
        def __init__(self, tech = 65):
        if tech ==65:
            self.testbench = home_address + '/Netlists/VCO_testbenchstatic_TT.scs'
            self.testfolder = home_address + '/Garbage/TrashVCO_1_1'
            self.minpar  = np.array([60e-9 ,0.2e-6 ,2  ,0.2e-6,2  ,2000 ,0.9  ,1.0 ])
            self.maxpar  = np.array([60e-9 ,1.2e-6 ,20 ,1.2e-6,20 ,2000 ,0.9  ,1.0 ])
            self.stppar  = np.array([1e-9  ,10e-9  ,1  ,10e-9 ,1  ,1    ,0.1  ,0.1 ])
            self.parname = ['lastt','wnnn','fnnn','wpppn','fppp', 'rres','VBIAS','VDD']
            self.metricname =
['power','vcm','vfs','fnoise','f1','f2','f3','f4','f5','f6','f7','f8']
            self.par_line_number = 7
            self.lst_metrics=[{'read':'c','filename':self.testfolder +
'/test.out/test.measure','number':2,'measurerange':range(9,10)},
{'read':'c','filename':self.testfolder +
'/test_cont_vout1.mt0','number':3,'measurerange':range(4,600)},
{'read':'c','filename':self.testfolder +
'/test_cont_tout1.mt0','number':3,'measurerange':range(4,600)}]
            self.runspectre1=TestSpice(simulator='afs',dict_folder={'testbench':self.testbench,
'trashfolder':self.testfolder},verbose = True)
```

Primitive function 1. Initialization (Example)

```
self.runspectre1 = TestSpice(simulator='afs',          # Simulator Type
        dict_folder={'testbench':self.testbench,       # Testbench netlist, COPY
        'trashfolder':self.testfolder},                # temp folder
        verbose = True)                                # True to see simulation success
```

Simulator Type:

1. ‘spectre’
2. ‘afs’
3. ‘apsplus’
4. ‘apsplusplus’
5. ‘spectre_ascii’
6. ‘afs_ascii’
7. ‘apsplus_ascii’
8. ‘apsplusplus_ascii’

Primitive function 2. Run SPICE

```
def normal_run(self, param):
        self.dict_parameters = YOUR_PARAMETER_DICTIONARY(param)
        error_occured = self.runspectre1.runspectre(self.dict_parameters)
                if error_occured:
                        out = []
                else:
                        out = self.runspectre1.readmetrics(self.lst_metrics)
         return out
```

Primitive function 2. Run SPICE (Example)

```
def normal_run(self,param):
        self.dict_parameters =
{'line_number':self.par_line_number,'name_params':self.parname,'value_params'
:param}
        error_occured = self.runspectre1.runspectre(self.dict_parameters)
        if error_occured:
            out1=[]
        else:
            out1 = self.runspectre1.readmetrics(lst_metrics=self.lst_metrics)
return out1
```

Primitive function 3. Metrics calculation

```
def analysis(self, out):
        metrics = YOURFUNC(out)
        return metrics
```

Primitive function 3. Metrics calculation (Example)

```
def analysis(self,lst_out):
        time_edges=np.array(lst_out[2])
        v_edges=np.array(lst_out[1])
        prd_edge=(time_edges[wside:]-time_edges[:-wside])/wside
        v_edges=v_edges[:-wside]
        v_edges=v_edges[lside:-rside]
        prd_edge=prd_edge[lside:-rside]
        x=v_edges
        y=1/prd_edge
        cof = np.polyfit(x, y, polynum)
        vcm=(v_edges[0] + v_edges[-1])/2
        vfs=(v_edges[-1]- v_edges[0] )/2
        vs = np.linspace(vcm-vfs,vcm+vfs,8)
        fout=cof[7]+cof[6]*vs
+cof[5]*vs**2+cof[4]*vs**3+cof[3]*vs**4+cof[2]*vs**5+cof[1]*vs**6+cof[0]*vs**
7
        return [vcm,vfs]+list(fout)
```

Primitive function 4. Run Spectre + Metrics calculation

```
def wholerun_normal(self,param):
        x = self.normal_run(param)
        w = self.analysis(x)
        return w
```

Primitive function 4. Run Spectre + Metrics calculation

```
def wholerun_normal(self,param):
        x = self.normal_run(param)
        w = self.analysis(x)
        return w
```

4 primitive functions:

```
__init__(tech = YOUR_TECHNOLOGY)

RAW_SPICE_DATA = normal_run(param = TESTBENCH_PARAMETERS)

METRICS = analysis(lst_out = RAW_SPICE_DATA)

METRICS = wholerun_normal(param = TESTBENCH_PARAMETERS)
```

After making the primitive functions you have access to secondary functions:

```
paramset(xmin,xmax,xstp)       # set the bounds: max, min, and step
PARAMETERS = random_param()    # randomly choose parameters within bounds
RAW_SPICE_DATA = random_run() # Runs normal_run with random parameter
METRICS = wholerun_random()    # Runs wholerun_normal with random parameters
# BAD_PARAMS are parameters that won't match with the given step. Ex. fingers should be integers and we cannot give finger = 1.5. This is BAD_PARAMS
PARAMETERS = param_std(BAD_PARAMS) # Fixes BAD_NUMBERS
RAW_SPICE_DATA = standard_run(BAD_PARAMS) # runs param_std with normal_run
METRICS = wholerun_std(BAD_PARAMS) # runs param_std with wholerun_normal
# Making dataset:
put_on_csv(self,tedad=NUMBER_OF_SAMPLES,outcsv=OUTPUTFILE.CSV,do_header=True_or_False)
# do_header : True if you want to have header on dataset, False when outcsv already exists
```