CppSim and Ngspice Data Modules for Python

Michael H. Perrott
http://www.cppsim.com/download_hspice_tools.html

October 25, 2015

The CppSim and Ngspice Data modules for Python include the classes CppSimData and NgspiceData, respectively, and additional functions to allow easy post-processing of CppSim and Ngspice simulation data using Python. This package is analagous to the Hspice Toolbox for Matlab/Octave, which allows easy post-processing of CppSim and Ngspice simulation data using Matlab and Octave.

We will begin this document by explaining how to set up the CppSim or Ngspice module for use with Python, and will then highlight key commands used within example Python scripts that illustrates loading of CppSim and Ngspice simulation data into Numpy arrays.

Setup

The CppSim and Ngspice Data modules are part of the main CppSim distribution, but are also provided as a standalone tar file at www.cppsim.com. It is recommended that you download and install the newest version of CppSim such that no further installation steps are required to include the CppSim and Ngspice Data modules. However, you can also download the file `cppsimdata_for_python.tar.gz` from the www.cppsim.com website, and then extract it within the CppSimShared directory of the CppSim installation. Once extracted, a Python directory should then appear within the CppSimShared

directory. We will refer to this `CppSimShared/Python` directory in the discussion to follow.

For Windows and Linux computers, both 32-bit and 64-bit versions of Python are supported. For Mac computers, it is assumed that you are running a 64-bit version of Python. In all cases, it is highly recommended to install the Express (i.e., free) version of the Enthought distribution of Python (www.enthought.com) to try out examples of using the CppSimData and NgspiceData modules. If you desire to use a 32-bit version of Python on a Mac, you will need to examine the `README` file within the `CppSimShared/Python` directory for instructions on how to compile the cppsimdata_lib.c and ngspicedata_lib.c files contained in that directory. Once you have compiled these files, you should then place the newly created `cppsimdata_lib.so` file and `ngspicedata_lib.so` file into the `macosx` subdirectory.

When running Python scripts that use the CppSim or Ngspice Data module, you should include the following lines at the top of such scripts:

```
# import cppsimdata and ngspicedata modules
import os
import sys
cppsimsharedhome = os.getenv("CPPSIMSHAREDHOME")
if cppsimsharedhome != None:
   CPPSIMSHARED_PATH = '%s' % cppsimsharedhome
else:
   home_dir = os.getenv("HOME")
   CPPSIMSHARED_PATH = '%s/CppSim/CppSimShared' % home_dir
sys.path.append(CPPSIMSHARED_PATH + '/Python')
from cppsimdata import *
from ngspicedata import *
```

The above lines direct Python to look in the appropriate directory when importing the CppSimData and NgspiceData modules. The last two lines import the CppSim and Ngspice Data modules for use within the given Python script.

CppSimData and NgspiceData Classes

The CppSimData and NgspiceData classes both include the following methods:

- `loadsig(filename)`

  - Loads the CppSim or Ngspice simulation signals within file `filename` into the CppSimData or NgspiceData object, respectively.

- `lssig()`
  - Returns a list of the signal names in the CppSimData/NgspiceData object. Note that `lssig('print')` can be used to print the signal names in addition to returning the list of signal names.

- `evalsig(nodename)`
  - Pulls out the data for signal `nodename` from the CppSimData/NgspiceData object and places into a Numpy array.

- `get_num_samples()`
  - Returns the number of samples for each signal contained in the CppSimData/NgspiceData object.

- `get_num_sigs()`
  - Returns the number of signals contained in the CppSimData/NgspiceData object.

- `get_filename()`
  - Returns the name of the file associated with the data contained in the CppSimData/NgspiceData object.

Example

The following commands are part of the `test_cppsimdata.py` file included in the `CppSimShared/Python` directory. Within Python, `cd` to that directory and then enter

```
%run test_cppsimdata.py
```

to see the results. Some key commands from this file are:

```
from pylab import *
from cppsimdata import *
data = CppSimData()
```

```
data.loadsig('test.tr0')
t = data.evalsig('TIME')
vin = data.evalsig('vin')
```

The first and second lines import the pylab routines (which include Numpy arrays) and the CppSimData module. The third line creates the CppSimData object, which is named `data`. Note that you can also load the CppSim simulation file as you create the CppSimData object by specifying the filename, such as `data = CppSimData('test.tr0')`. The fourth line loads the CppSim simulation data from file `test.tr0` into the CppSimData object using the `loadsig` method. The fifth and sixth lines transfer the data corresponding to signals `TIME` and `vin` into Numpy arrays `t` and `vin`, respectively. From there, the Numpy arrays can be plotted or used for post-processing operations.

Note that using the NgspiceData class is very similar to the above example, with the main difference being that an Ngspice raw file is loaded and the following commands are used instead of lines two through four above:

```
from ngspicedata import *
data = NgspiceData()
data.loadsig('simrun.raw')
```

CppSimData Functions

A few other functions are available as part of the CppSim Data module:

- `cppsim(sim_file)`
  - Allows CppSim to be run directly from Python. If `sim_file` is not specified (i.e., `cppsim()` is run), it is assumed to be `test.par`.
- `calc_pll_phasenoise(noiseout,Ts)`
  - Returns `f` (Hz) and `Pxx_db` (dBc/Hz) given the `noiseout` signal and time step, `Ts`, of the `noiseout` samples.

Example

To see the `calc_pll_phasenoise(noiseout,Ts)` in action, an example Python script has been provided. Within Python, `cd` to the `CppSimShared/Python` directory and

then enter:

```
%run test_phase_noise_plot.py
```

A phase noise plot should appear. For further details, use an editor to examine the contents of the `test_phase_noise_plot.py` file.

NgspiceData Functions

A few other functions are available as part of the Ngspice Data module:

- `ngsim(hspc_file)`
  - Allows Ngspice to be run directly from Python. If `hpsc_file` is not specified (i.e., `ngsim()` is run), it is assumed to be `test.hspc`.
- `hspc_set_param(param_name, param_value, hspc_file)`
  - Changes the parameter value of `param_name` to `param_value` within the corresponding `.param` line of the specified `hspc_file`. This function facilitates parametric sweeps when running Ngspice from within Python using the `ngsim()` command.
- `hspc_addline(new_line, hspc_file)`
  - Adds the line `new_line` to the specified `hspc_file`. This function facilitates alter runs and parametric sweeps when running Ngspice from within Python using the `ngsim()` command.
- `hspc_addline_continued(new_line, hspc_file)`
  - Adds the line `new_line` to the specified `hspc_file`. This function facilitates alter runs and parametric sweeps when running Ngspice from within Python using the `ngsim()` command. Note that `hspc_addline()` is used to create one line, and `hspc_addline_continued()` is used to create additional lines after `hspc_addline()` has been run.
- `eyesig(period, start_off, time, data)`
  - Plots the eyedigram for signal `data` given its associated time signal `time` and the specified `period` and starting time offset, `start_off`.

Example

Please see the section `Running Parameter Sweeps using Python Scripting` in the PDF document `NGspice Primer Within CppSim (Version 5) Framework` available at `http://www.cppsim.com/manuals.html`.