**How to PySPICE**

2018/02/07

Steve Wood

PySPICE is a new Python library that allows for true mixed-signal simulation using Python for the digital portion of the circuit, and NGSPICE for the analog portion. This is not possible in LTSPICE. In LTSPICE, any simulation of an MCU or FPGA is limited to what can be built using logic gates, analog circuitry, and behavioral sources. This is tedious at best for any significant amount of control logic.

Python is not C, but there are ways to wrap C in Python beyond the scope of this document. Thus it may be possible to take your actual MCU C code, wrap it in Python, and use it in PySPICE. For an FPGA based design, you can simply write your FPGA code in MyHDL, and it ***is*** Python already. From the MyHDL code, you can press a button and get synthesizeable code in either Verilog or VHDL.

<http://myhdl.org/>

Is PySPICE fast enough to be practical? Yes. On my computer with my example I can run a BLDC motor at a bit faster than 1ms simulation time per second real time. Also, the PySPICE version runs slightly faster than the the LTSPICE version.

Being a recent open-source development, there are not too many mixed-signal PySPICE examples available. I found only one, and it was pretty limited. Thus, I recommend using what I’ve prepared as a more comprehensive example of what PySPICE can do in terms of mixed-signal simulation.

**Installation:**

You'll need to install Python3.6, PySpice, NGSPICE, Scipy, Numpy, and MatPlotLib (be sure to use all Python3 versions; use the “pip3” command instead of “pip”).

Link to PySpice:

<https://pyspice.fabrice-salvaire.fr/>

(ignore the warning you might get on Firefox – the site is legitimate)

Link to NGSPICE:

<http://ngspice.sourceforge.net/>

Link to Scipy, Numpy, and Matplotlib:

<https://www.scipy.org/>

 If you prefer, all three of the above are included in the Anaconda package:

<https://www.anaconda.com/download/>

Note: I was successful installing Anaconda on Windows 7, but I had problems installing it on Ubuntu 16.4 on three different machines. Anaconda also has a lot of extra stuff not needed for PySpice, so I would recommend “a la carte” for any Linux installation.

**Usage:**

NGSPICE has no schematic capture, so you can either enter your netlist by hand (tedious) or use some other schematic capture. If a SPICE netlist is provided, the netlist will be parsed to convert it to Python in the PySpice format. The Python can be entered directly, but it’s much faster to generate a SPICE netlist with a schematic capture tool, and then let PySpice parse it. LTSPICE or KiCAD can be used.

It’s advantageous to use LTSPICE if you have some way to test your netlist in LTSPICE before porting it over to PySpice. Simulation is also available through KiCAD, but I’ve not tried it yet. In my example, a crude version of the logic controller was implemented in LTSPICE to allow for debug prior to the port to PySpice. Unfortunately LTSPICE and NGSPICE do not have exactly the same syntax; however, it is possible to capture a schematic in LTSPICE with a resulting netlist that will run in both LTSPICE and NGSPICE if the following rules are observed:

Setting up Netlist:

\* No utf-8 "mu" for micro allowed. Must use simple 'u'.

\* No spaces allowed in equations.

\* No IF statements (they exist in NGspice, but the format is different).

\* No "limit" function. Use min(max(,),) instead.

\* No .param statements; parameters must be passed at the instance call out only.

\* No .ic statements ; initial conditions can only exist at the instance call out and in Python.

\* No SGN function. Check NGspice manual for other functions that differ from Ltspice.

\* Only independent sources can be referenced for currents used in expressions (no i(L4) allowed for example; instead insert a 0V source and read the current through it).

\* Do not rely on case to differentiate labels. Netlist is converted to all lower case by PySpice.

\* Need to prepend "dc x " (where 'x' is the initial value, usually zero) manually for all sources that do not have an explicit DC value (PWL, PULSE, etc), otherwise NGspice will warn that the time 0 value is being used for the DC solution (benign warning, but best to clean it up).

\* Inductors and capacitors need to have their parasitics shown explicitly, not built into the component.

\* No dashes allowed in file names.

\* The top level subcircuit of the hierarchy should have no parameters (if you have to add a dummy level, then do so). It is fine if your LTSPICE testbench has parameters, as it will get deleted anyway. The reason you need this top level subcircuit is so that the other models in the netlist can get called.

\* It is the nets from the python code that get passed to and from the nets in the netlist file. That is what must be called out in the subcircuits and linked to the proper nets in the python file

There are probably other rules that will need to be added to this list. These are just the ones I discovered the hard way on my particular circuit. Note that this list goes both ways. There are some things NGSPICE can do that LTSPICE cannot (% for modulo arithmetic, for example).

Determine the minimum frequency needed for your MCU or FPGA controller. Place a simple pulse generator running at this frequency on your LTSPICE schematic that outputs to the very top of the hierarchy. Use zero rise and fall time for this clock (normally ill-advised for convergence reasons in SPICE, but this clock isn’t going anywhere but to Python-land). Since it goes nowhere, it won’t slow down your LTSPICE simulation very much.

Once you’ve entered your schematic you’ll want to “View” “SPICE Netlist” from the top level of your schematic and copy everything in the window to the clipboard. Paste it into your favorite text editor (I use Geany) and name the file “ngspice\_top.lib” (if you want it to work with my example). You can also just find the LTSPICE netlist file in the appropriate directory. Next you’ll need to do same manual edits. These can be scripted later if desired:

\* Remove any LTSPICE *testbench* subcircuit (this function will be performed by Python; in the example it’s “control\_cont\_nohalladv” that gets removed)

\* Delete everything that's not a subcircuit (including end of file top-level statements such as .tran, .end); This file should be only a collection of subcircuits, not a complete SPICE netlist. Comments are fine.

\* Add parameters to each subckt line (just copy from any instantiation of said subcircuit; do not include "params:", but just a space and then a space delimited set of parameters)

\* Change any unicode 'mu' to 'u'

\* Add top subcircuit ngspice\_top using instantiation line for XX1 in original top-level netlist (see example)

**Next, the Python code**. I would recommend just copying my example “MS\_USA13030-X1\_Cont.py” and changing it to suit your needs.

Initialization: super().\_\_init\_\_(\*\*kwargs)

The first lines of code are initialization. The first initialization section is just there to prevent a benign error when the simulation starts. If you see “"AttributeError: 'MyNgSpiceShared' object has no attribute 'xxxxxx'" at the beginning of a simulation run, then a variable needs to be initialized. The missing attribute soon is generated anyway, even without initialization, so this error is benign. The second section of initialization is actually needed to prevent fatal errors.

def get\_vsrc\_data(...)

Next is “def get\_vsrc\_data(...)”. This function provides digital signals from the Python block to the NGSPICE block. It also processes analog signals coming from the NGSPICE block. One line of code needs some explanation I think: “if (round(self.clk\_out) != round(self.clk\_out\_old)) and (time != self.time\_filt): #This "if" statement evaluated once for each node for each NgSpice timestep; for speed, the calculations following this conditional are only executed once per dig\_timestep.”

PySpice calls up the controller Python (the digital block) for every top-level node for every NGSPICE timestep. Typically there will be far more NGSPICE timesteps than clock edges, so some simulation efficiency can be gained by screening out those timesteps that do not coincide with (or technically, fall right after) a digital clock edge. There are some situations where every timestep must be considered by the digital block, or at least part of it. For example, if you have to asynchronously capture an analog edge that might come and go between digital clock edges. In my example, there was no need for looking at any timesteps other than those corresponding with clock edges.

In addition to skipping the non-clock-edge timesteps, for most of the digital block we can skip all but one of the nodes (since every node calls up the Python digital block for each timestep). In the example, it doesn’t really matter which node is the “trigger” for the execution, so I use the first one that occurs. The conditional based on “ time != self.time\_filt” screens all but the first call of the digital block per timestep.

The main portion in the code that follows is the control algorithm. Note that any variable that must be kept in memory between function calls needs to be appended with “self.”u, whereas variables that don’t need to be retained don’t need the prefix.

Outside the “if” detailed above are the tests to see which node is responsible for the function call. See below the comment, “Outputs below go from Python to NGspice.” The node responsible for the function call determines the node value that needs to be assigned to “voltage[0]” for this particular function call. The “Dummy outputs” section contains those nodes that have been brought out for probing from the digital block. Technically they are no different than the nodes above, except that they don’t actually connect to anything in the analog block.

The “# NGspice data sent to Python” section contains all signals from the analog block being sent to the digital block. I chose to append all signal names in this block with “\_out”, but the naming is arbitrary. It is possible to send “buried” or non-top-level signals. A commented-out example is given for extracting a buried node.

Netlist:

Next is just the path for any subcircuit libraries, which must all end in “.lib”. It appears that all \*.lib in the specified path will be parsed, so you’ll want to keep only what’s needed in the path. With this in mind, I’m not sure why the “circuit.include(spice\_library['ngspice\_top'])” line is necessary. I’ve not experimented with this.

In my example, most all of the netlist is provided as SPICE subcircuits to be parsed; however, a very small portion of the netlist is written in Python. The “circuit.” lines comprise this netlist. These lines provide the sources in the SPICE netlist driven by the digital block and call out the SPICE subcircuit to be parsed (ngspice\_top with reference designator X1).

The next section of code assigns variables that get passed with “ngspice\_shared”. These are all just parameters affecting the control algorithm or simulation time.

The simulator is called in the “shared” mode. This mode is for mixed-signal simulation (see PySpice documentation). All NGSPICE parameters and options are available. Note that initial conditions should be called from the very top level in Python. There may be a way to assign these in the NGSPICE netlist, but I’ve not had success with it yet.

I think it should be also possible to assign NGSPICE parameters from Python, but I was unable to do so. I assigned all parameters in SPICE, and my top-level subcircuit contains no parameters.

The “sim\_start\_time” and “sim\_end\_time” is just something I added because it’s handy for benchmarking simulations.

The “# PLOTTING RESULTS:” section that follows employs Matplotlib for plotting. Examples are given for plotting both top level and hierarchically “buried” signals.

With each run, a “pickle” dump is made into “myplot.pkl” in the working directory. If you run “plot\_olddata.py”, it will read the pkl file and duplicate the graph shown originally. There is a bug though. The “xshared” option to allow all subplots to be zoomed along the x-axis together is not working properly. Zoom works, but each subplot will have an independent x-axis (very inconvenient for zooming).

8/5/19, mq: Setting ngspice library path:

<https://github.com/FabriceSalvaire/PySpice/issues/95>

Python can have difficulty finding the library path when running python 32bit and the file is meant for 64 bit.

To plot signals from python block in NGspice plot:

1. Include if statement in section for outputs from python to ng spice:

elif node == 'vb':

voltage[0]=self.vb\_val

1. Include the analog stimulus line in NGspice block:

circuit.V('a', 'a', circuit.gnd, 'dc 0 external')

1. Include the plot function in the plotting block:

plot(analysis.va)