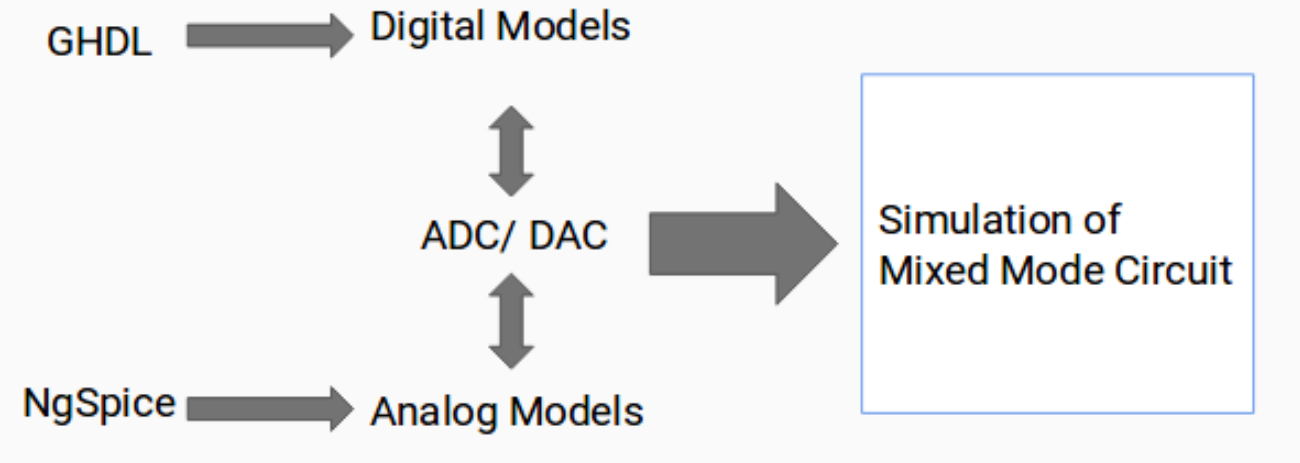
NGHDL OVERVIEW :



What exactly interfacing of ngspice ghdl do?

→ Ngspice support mixed mode simulation. It can simulate both digital and analog component.

→ Ngspice has something called model which define the functionality of your circuit, which can be used in the netlist.

→ For example you can create adder model in ngspice and use it in any circuit netlist of ngspice.

Now the question is if we already have digital model in ngspice why this interfacing ?

→ Well in ngspice it is little tediouse to write your digital model. But many people are familiar with ghdl and can easily write the vhdl code.

→ So the idea of interfacing is just to write ghdl code for a model and install it as dummy model in ngspice. So whenever ngspice look for that model it will actually call the ghdl to get the result.

Pre-requisites -

→ *Ubuntu 16.04* - You can try it on other version and let us know

→ *Python 2.7*

→ *PyQt4*

→ *Ghdl* - Since PPA and package not available, needed to build from source using LLVM backend

→ *NgSpice* - Need to add custom models to ngspice using xspice, so need to build from source

How to install?

1. Clone this repository.
2. Run `./install-nghdl.sh --install` It will install ngspice from source code and put it in $HOME.

Few words about installed code structure -

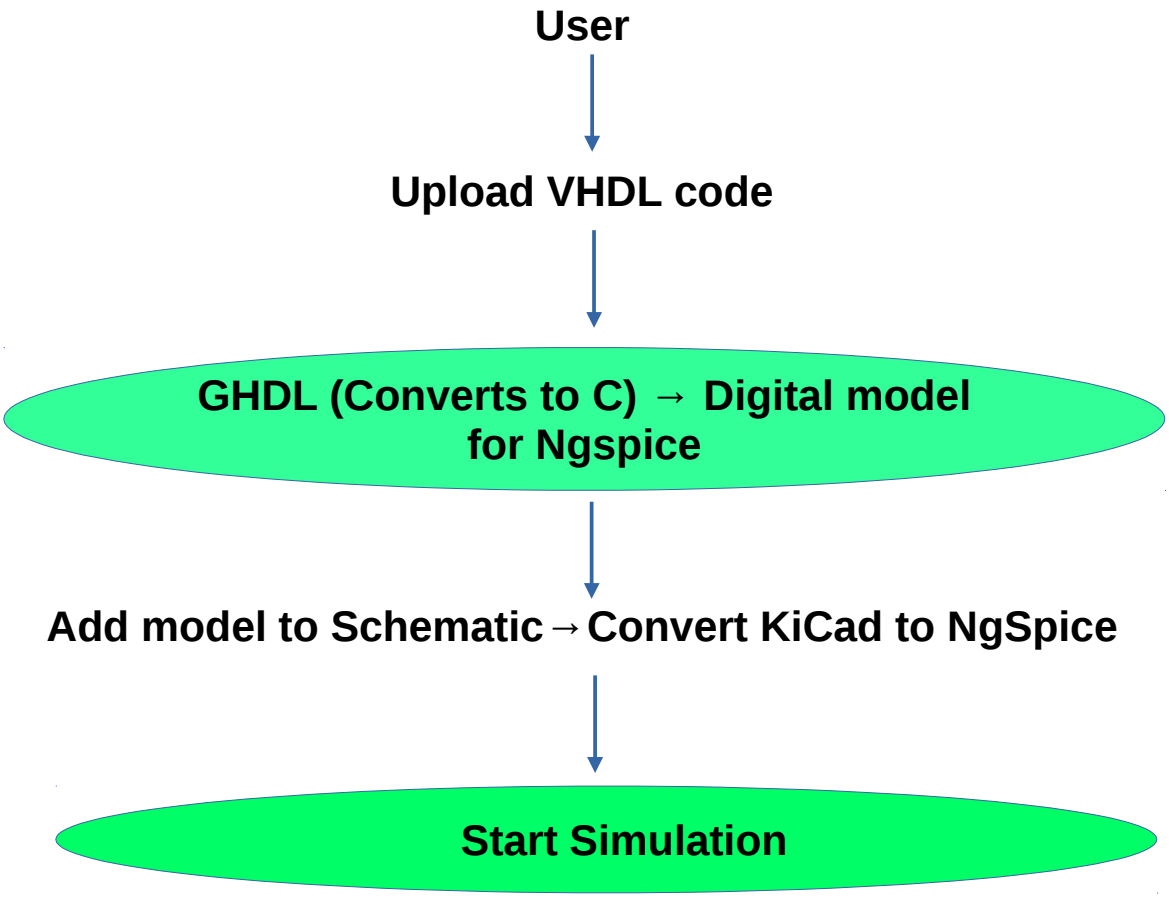
→ Ngspice will be installed in home directory $HOME. If you already have ngspice-26 directory there it will take its backup.

→ Source code for all other file will be present in ~/.esim-nghdl

→ symlink nghdl is stored in /usr/local/bin

NGHDL WORKING :

1. Run nghdl in command terminal.
2. Upload your vhdl file.
3. Model will be created with your name of your vhdl file.
4. You can use this model in your netlist.



1. Digital model for NgSpice (from VHDL) :

Creates a dummy model for ngspice and Generation of C files for simulation :

a. Model directory created
b. Model added to modpath
c. Model files generated as follows -
   i. Cfunc.mod → Defines NgSpice Model (Converted to C)
   ii. ifspecs.ifs → Defines NgSpice Model (Converted to C)
   iii. Nghdl server script (start_server.sh)
   iv. sock_pkg.vhdl

d. Runs "*make*" and "*make install*" to generate code-models for NgSpice

The above mentioned task is done by two code files in ***eSim-1.1.2*** --> ***nghdl*** --> ***src***:

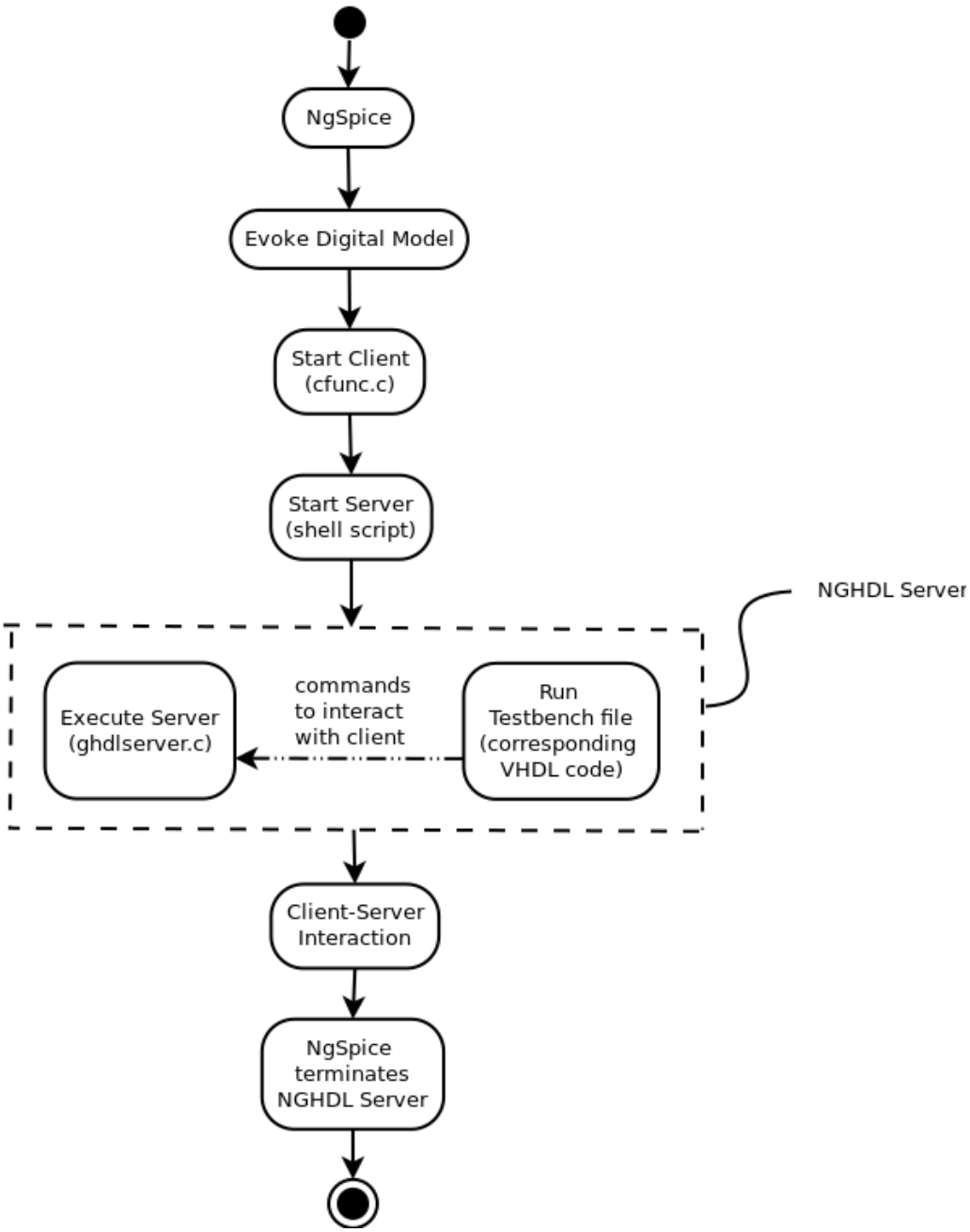
a. **model-generation.py**
b. **ngspice_ghdl.py**

The **Interface Specification File** is a text file that describes, in a tabular format, information needed for the code model to be properly interpreted by the simulator when it is placed with other circuit components into a SPICE deck. This information includes such things as the parameter names, parameter default values, and the name of the model itself. The specific format presented to you in the Interface Specification File template must be followed exactly, but is quite straight forward.

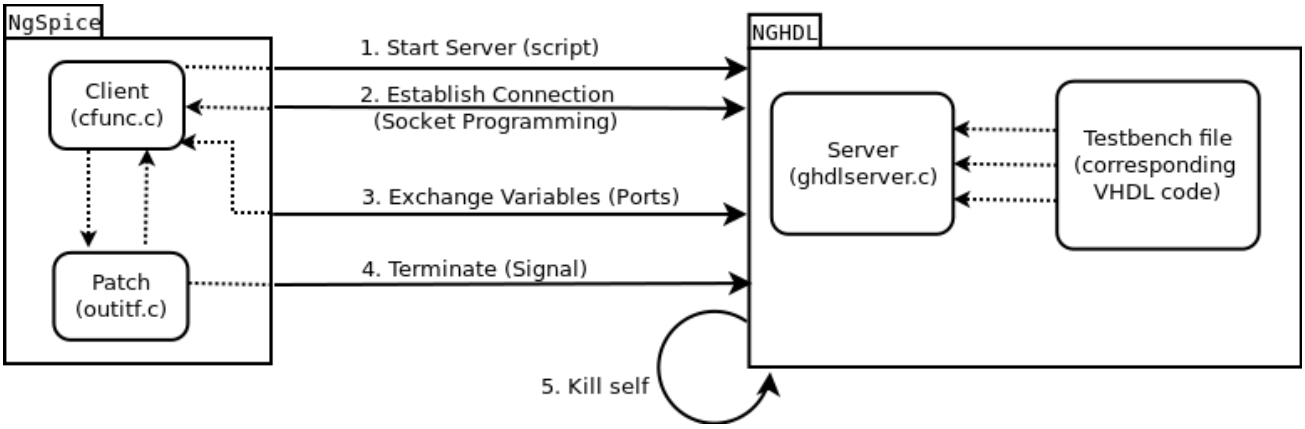
The **Model Definition File** contains a C programming language function definition. This function specifies the operations to be performed within the model on the data passed to it by the simulator. Special macros are provided that allow the function to retrieve input data and return output data. Similarly, macros are provided to allow for such things as storage of information between iteration time-points and sending of error messages.

2. NgSpice Simulation :

   a. NgSpice runs *"cfunc.c"* of the respective model which acts as a client.
   b. This client starts server shell script (***start_server.sh***) which executes Testbench file linked with the Server file (***ghdlserver.c***).
   c. Testbench acts as a controller of NGHDL Server and commands the server file to connect and interact with the client.
   d. NgSpice sends a signal to the Server file to terminate.
   e. The Server file, in turn, terminates itself due to which Testbench also gets closed.



Client – Server Interaction :



NGSPICE :

1. Analog Simulation -

The circuit response is obtained by iteratively solving Kirchhoff's Laws for the circuit at time steps selected to ensure the solution has converged to a stable value and that numerical approximations of integrations are sufficiently accurate. Since Kirchhoff's laws form a set of simultaneous equations, the simulator operates by solving a matrix of equations at each time point. This matrix processing generally results in slower simulation times when compared to digital circuit simulators.

2. Digital Simulation -

When an event occurs, the simulator examines only those circuit elements that are affected by the event. As a result, matrix analysis is not required in digital simulators. By comparison, analog simulators must iteratively solve for the behavior of the entire circuit because of the forward and reverse transmission properties of analog components. This difference results in a considerable computational advantage for digital circuit simulators, which is reflected in the significantly greater speed of digital simulations.

3. Mixed-Mode Simulation -

Glued mode simulators actually use two simulators, one of which is analog and the other digital. This type of simulator must define an input/output protocol so that the two executables can communicate with each other effectively. The communication constraints tend to reduce the speed, and sometimes the accuracy, of the complete simulator.

On the other hand, the use of a glued mode simulator allows the component models developed for the separate executables to be used without modification.

4. DC Analysis -

For DC analysis, the time-varying behavior of reactive elements is neglected and the simulator calculates the DC solution of the circuit.

The simulator algorithm subdivides the circuit into those portions which require the analog simulator algorithm and those which require the event-driven algorithm. Each subsystem block is then iterated to solution, with the interfaces between analog nodes and event-driven nodes iterated for consistency across the entire system.

5. Transient Analysis -

Transient analysis is an extension of DC analysis to the time domain. A transient analysis begins by obtaining a DC solution to provide a point of departure for simulating time-varying behavior. Once the DC solution is obtained, the time-dependent aspects of the system are reintroduced, and the two simulator algorithms incrementally solve for the time varying behavior of the entire system.

6. Convergence Failure -

Although the algorithm used in ngspice has been found to be very reliable, in some cases it fails to converge to a solution. When this failure occurs, the program terminates the job. Failure to converge in dc analysis is usually due to an error in specifying circuit connections, element values, or model parameter values. Regenerative switching circuits or **circuits with positive feedback probably will not converge in the dc analysis** unless the *OFF* option is used for some of the devices in the feedback path, *.nodeset* control line is used to force the circuit to converge to the desired state.