CppSim/VppSim Primer for Cadence
Version 5.1

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

August 5, 2012

Table of Contents

Introduction

CppSim is a general behavioral framework that leverages the C++ language to achieve very fast simulation times, and a graphical framework to allow ease of design entry and modification. Users may freely use this package for either educational or industrial purposes without restriction. However, the package and all of its components come with ***no warranty or support***.

The CppSim/VppSim framework is a superset of three different, but related, simulation environments:

- CppSim: C++ and Verilog co-simulation, with C++ providing the primary simulation engine.
- VppSim: C++ and Verilog co-simulation, with Verilog providing the primary simulation engine.
- AMS with VppSim modules: C++, Verilog, VHDL, and SPICE co-simulation

Of the above, CppSim is the most mature and has been widely used in the form of a freely available Windows package. The primary difference between this distribution and the Windows one is that the Linux platform is used, and Cadence is used for schematic entry. The VppSim simulator allows use of CppSim modules within the Icarus Verilog or Cadence ncverilog simulator. AMS with VppSim modules allows use of CppSim modules within the AMS Designer program.

The CppSim/VppSim framework has the advantage of allowing very sophisticated and fast simulation of complex mixed-signal circuits/systems at the behavioral level by using CppSim or VppSim within the Cadence design environment. The user can seamless move from analog behavioral level simulations using CppSim into digital design (i.e., Verilog) using VppSim. The user can also co-simulate the C++ behavior blocks with SPICE (and Verilog/VHDL) models by using AMS Designer in conjunction with VppSim modules.

Known Bugs

1) The AMS interface is quite limited at this stage. In particular, all VppSim modules are assumed to have a constant time step of #1 in the Verilog simulation engine.
2) Some of the CppSim examples have not been tested within the Cadence setup.

Setup

1) In your web browser, go to http://www.cppsim.com and look for the links corresponding to the following files:
   - **cppsim_dist5.tar.gz**
     - This tar file contains the entire CppSim/VppSim package for Cadence.
   - **amsd.tar.gz**
     - This tar file contains an example that shows how CppSim modules can be incorporated within AMS runs.
2) Download the file **cppsim_dist5.tar.gz**
3) At the Linux shell prompt, type
   > tar zxvf cppsim_dist5.tar.gz
   - This will untar the file and create a directory called **CppSim_Dist**
   - Within **CppSim_Dist**, you will see three directories and two files:

- **CppSim**
  - This directory is intended to be replicated for each individual user, and will likely be placed in the user's home directory. It contains files used for running individual simulations, and is the place where each user's simulation results will be stored.
- **CppSimShared**
  - This directory is intended to be shared by all users at a given site. It contains all supporting binaries, shared **Cadence** libraries, shared code, and shared Skill code for the **CppSim** GUI within **Cadence**. It also contains a directory **MatlabGui** within which the **cppsimview** waveform viewer code is located.
- **menus**
  - This directory is used to set up the **CppSim** GUI menu item within **Cadence** schematic windows.
- **cshrc_code**
  - This file contains example code that should be placed within the **.tcshrc.mine** or **.cshrc.mine** file of each user. If you are not using **tcsh** or **csh**, then you will need to change the commands in this file according to whatever Linux shell you use.
- **bashrc_code**
  - This file contains example code that should be placed within the **.bashrc** file of each user. This assumes you use the **bash** shell rather than the **csh** or **tcsh** shell.

4) Place each of the directories within **CppSim_Dist** at appropriate locations
   - **CppSim**: should be placed in the user's home directory or some other appropriate location
     - It will be assumed that it is placed in the user's home directory at **~/CppSim** for the remainder of this document.
   - **CppSimShared**: should be placed in a directory that is readily accessible by all users
     - It will be assumed that it is placed at **/u/CppSimShared** for the remainder of this document. (Note that it can also just be placed in the user's home directory if desired).
   - **menus**: this MUST be placed in the user's home directory in order for **Cadence** to recognize it.
5) If you want the AMS example, then also download **amsd.tar.gz**
   - At the Linux shell prompt, type
     > tar zxvf amsd.tar.gz
     - This will untar the file and create a directory called **AMSD**
     - Place the **AMSD** directory in the user's home directory or some other appropriate location. In the remainder of this document, it will be assumed to have been placed in the user's home directory at **~/AMSD**.
6) Add and appropriately modify the code in **cshrc_code (or bashrc_code)** to the user's **.tcshrc.mine** (or **.bashrc**) file or whatever the appropriate startup script is for their Linux shell environment. Be sure to source the startup script before running **Cadence**.
   - Make sure that your **EDITOR** environment variable is set to your desired graphical text editor (such as emacs or gvim)

Using CppSim within Cadence

Given completion of the above setup, we now explain the basics of running CppSim within Cadence.

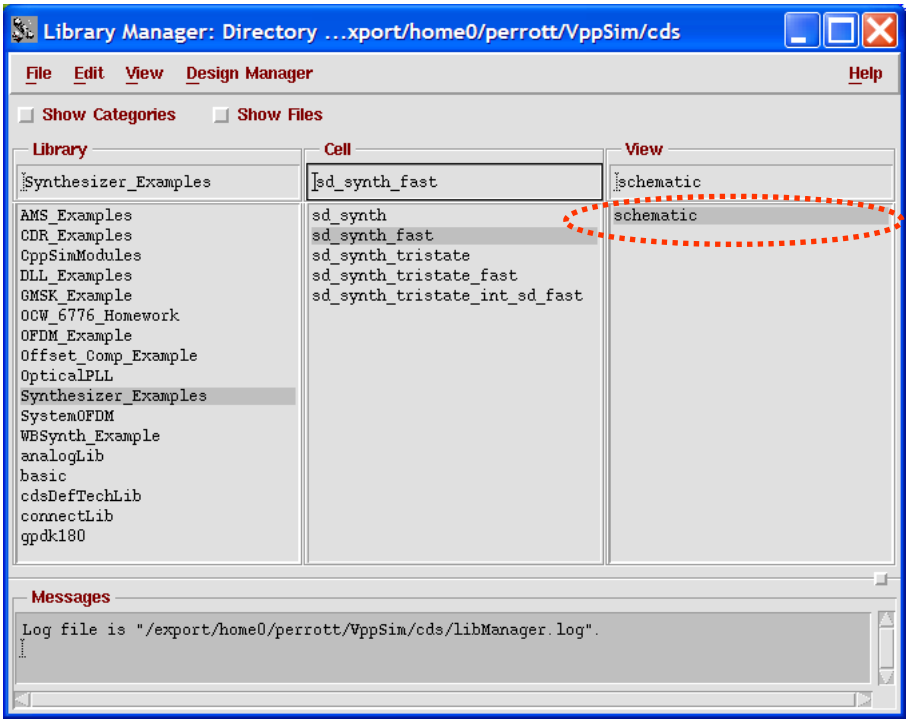
A. Starting Cadence

- Be sure to start Cadence within the **~/CppSim/cds** directory. This is necessary in order to access the appropriate **cds.lib** file and **.cdsinit file** available in that directory. Experienced Cadence users can move these files to a different directory of their choice, and start Cadence there. At the Linux prompt, type:

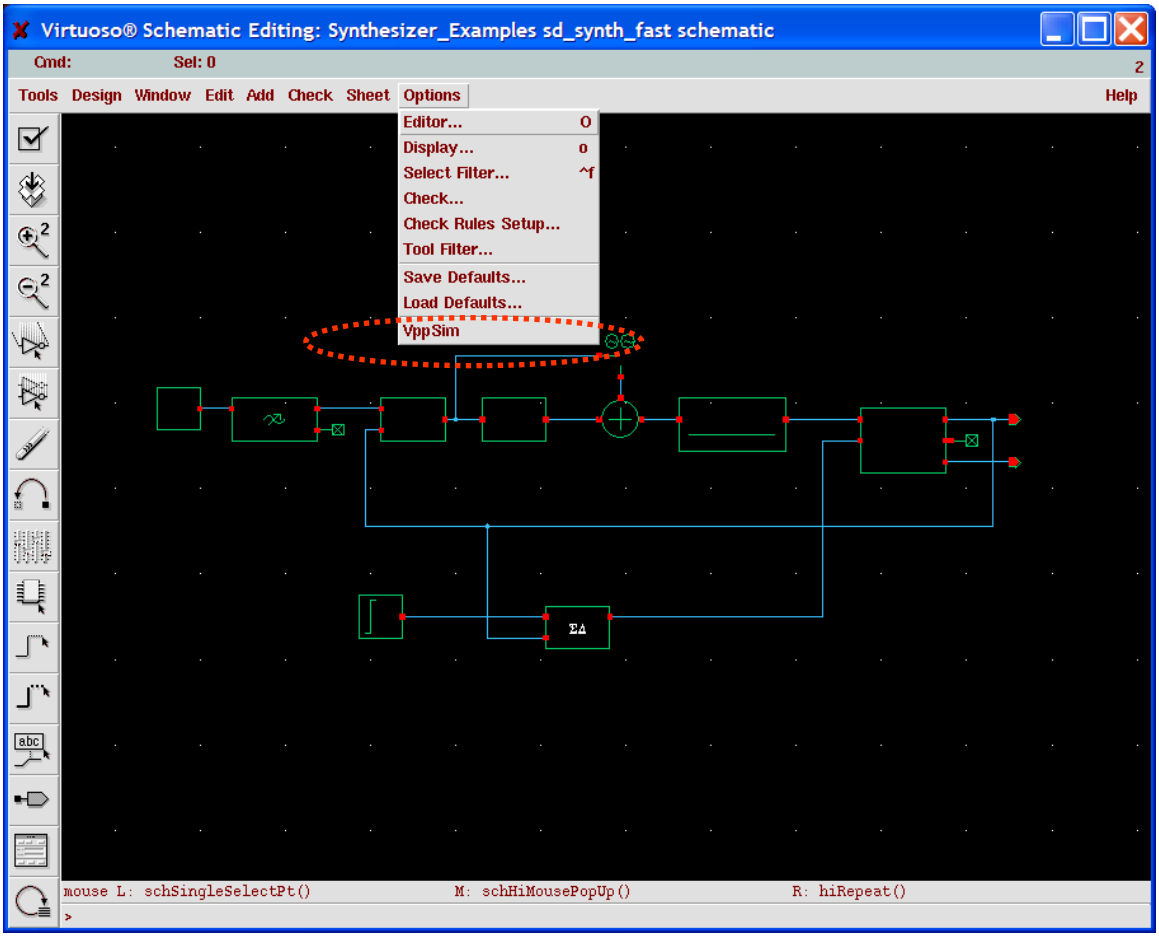
```
For Cadence 5:
> cd ~/CppSim/cds
> icms &

For Cadence 6:
> cd ~/CppSim/cds
> virtuoso &
```

B. CppSim simulation example

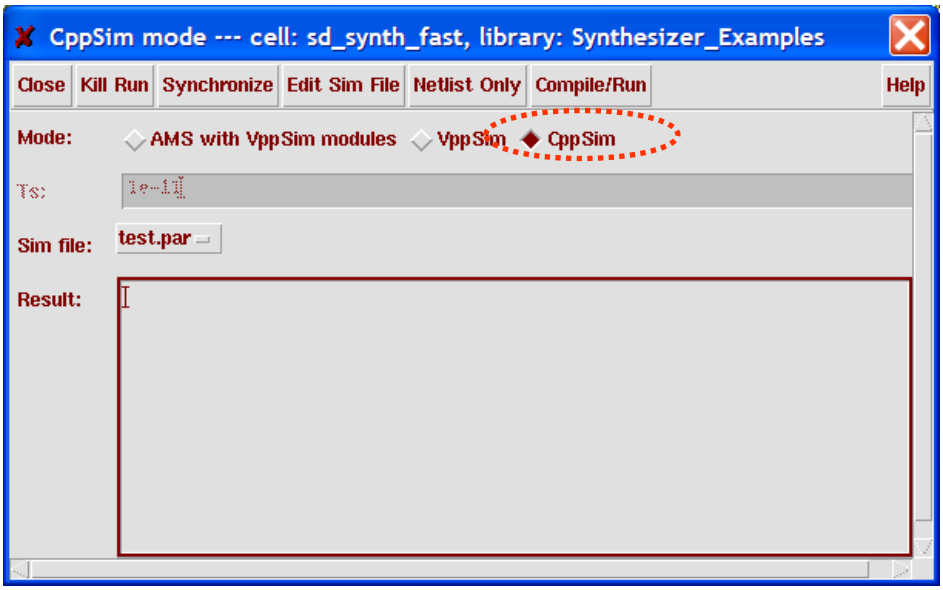
- Open the **sd_synth_fast** schematic cellview within the **Synthesizer_Examples** library by double-clicking on it within the Cadence Library Manager.



- In the newly opened schematic windows, select **Options->CppSim/VppSim** to open up the **CppSim/VppSim GUI** within Cadence.

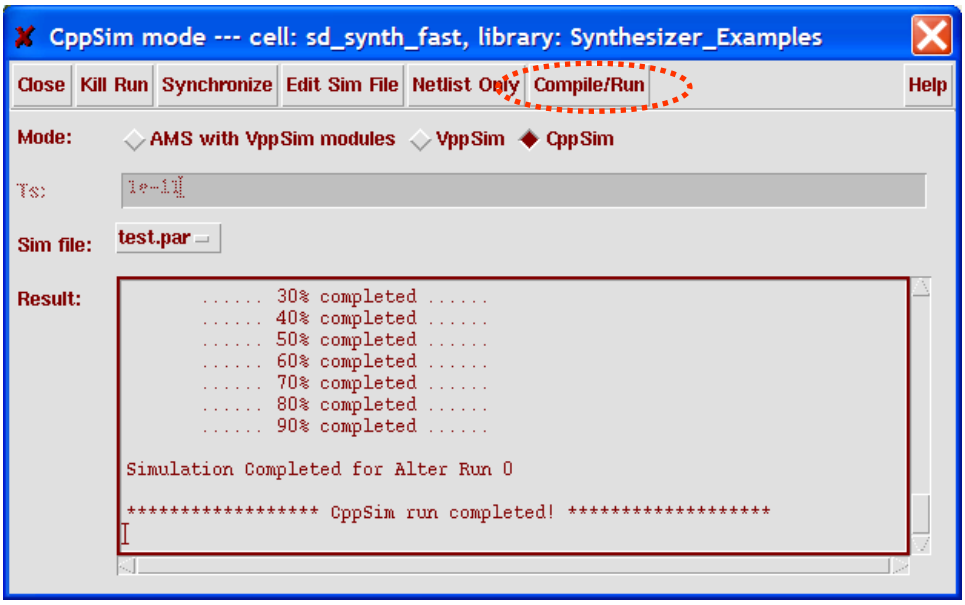


- Click on the **CppSim** radio button within the **CppSim/VppSim GUI**.



- Click on the **Compile/Run** button within the **CppSim/VppSim GUI**. The current schematic will be automatically netlisted and then the **CppSim** simulation will begin. You will see some warning messages, but the end result should be as shown below. Note:

expand the **CppSim/VppSim GUI** window in order to place the scrollbars within the **Result:** portion of the GUI as shown below.



C. Using CppSimView (the CppSim custom viewer)

There are two ways of running CppSimView in order to view simulation results: from within Matlab or from the Linux shell window. Note that there are some slight differences between CppSimView for the two approaches, and we will assume that it is run from within Matlab for the examples to follow.

Running CppSimView within Matlab

Running CppSimView from within Matlab has the advantage that it requires only minor setup effort, but carries the disadvantage of requiring Matlab and its Signal Processing Toolbox to be installed on the machine.

- Within Matlab (version 7 or later):

  Note that you must first compile two files in Matlab for cppsimview to work:
  > cd /u/CppSimShared/MatlabGui/CppSimView
  > mex loadsig_cppsim.c
  > mex min_max_decimate.c

  > addpath('/u/CppSimShared/MatlabGui/CppSimView')
  > cppsimview

Running CppSimView from the Linux shell

Running CppSimView from the Linux shell has the advantage of not requiring Matlab to be installed on the machine, but carries the disadvantage of requiring Wine to be installed. While Wine is free and readily available, its installation will likely require the help of a system administrator. Please see the

Wine webpage at http://www.winehq.org for further details on how to download and install this software.

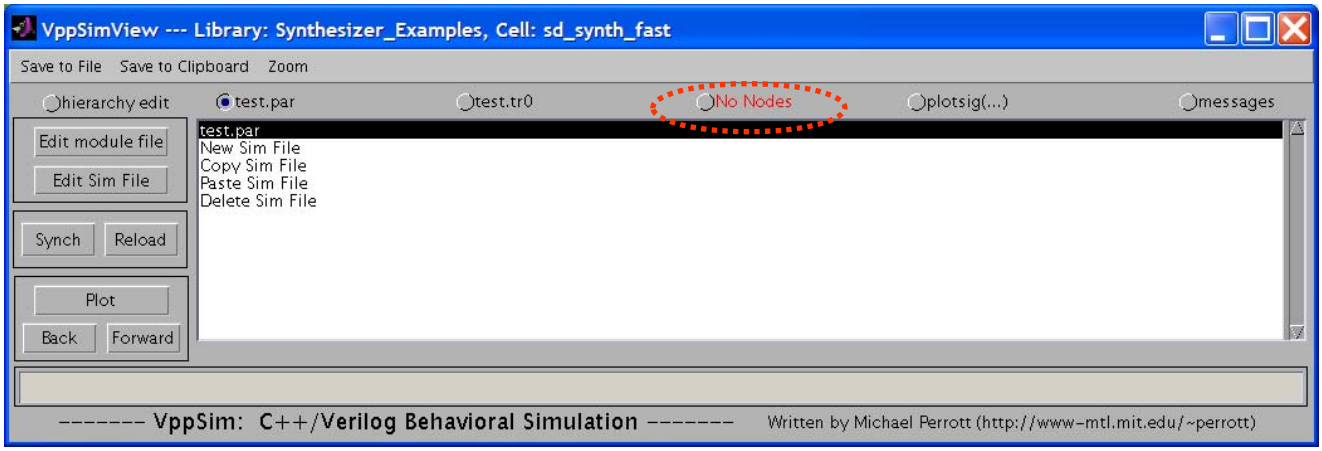
- At the Linux shell prompt:
  > cppsimview
  - Note that cppsimview will provide an error message if Wine is not installed.

As a side comment, note that the PLL Design Assistant can also be run from the Linux shell prompt once Wine is installed:

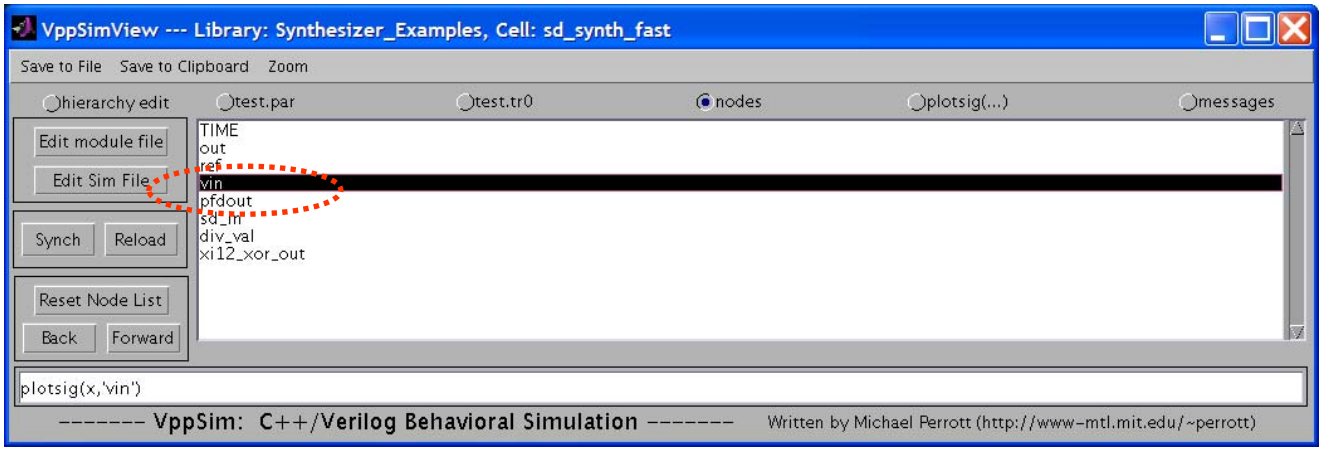
- At the Linux shell prompt:
  > plldesign

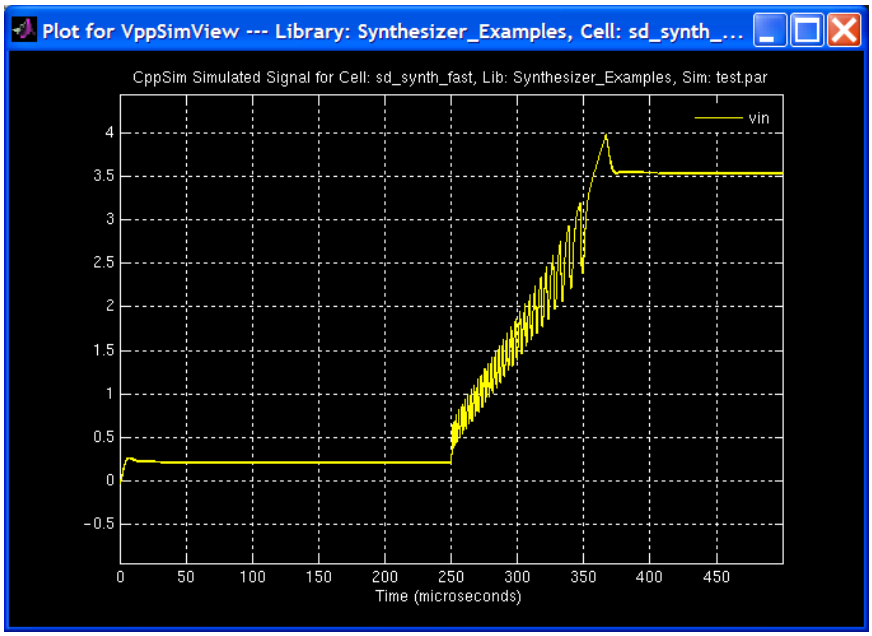
Using CppSimView (either by running in Matlab or from the Linux shell)

- The viewer should appear and will *automatically synch* to the last cellview that was *netlisted* from the **CppSim/VppSim GUI**. Such netlisting occurs whenever you click on the **Netlist Only** or **Compile/Run** button with **the CppSim/VppSim GUI**. Pushing **Synch** in the **cppsimview** window will also synchronize to the last cellview that was netlisted from the **CppSim/VppSim GUI**.
- Click on the **No Nodes** radio button in **cppsimview** in order to display the nodes that were probed in the CppSim output file **test.tr0**.



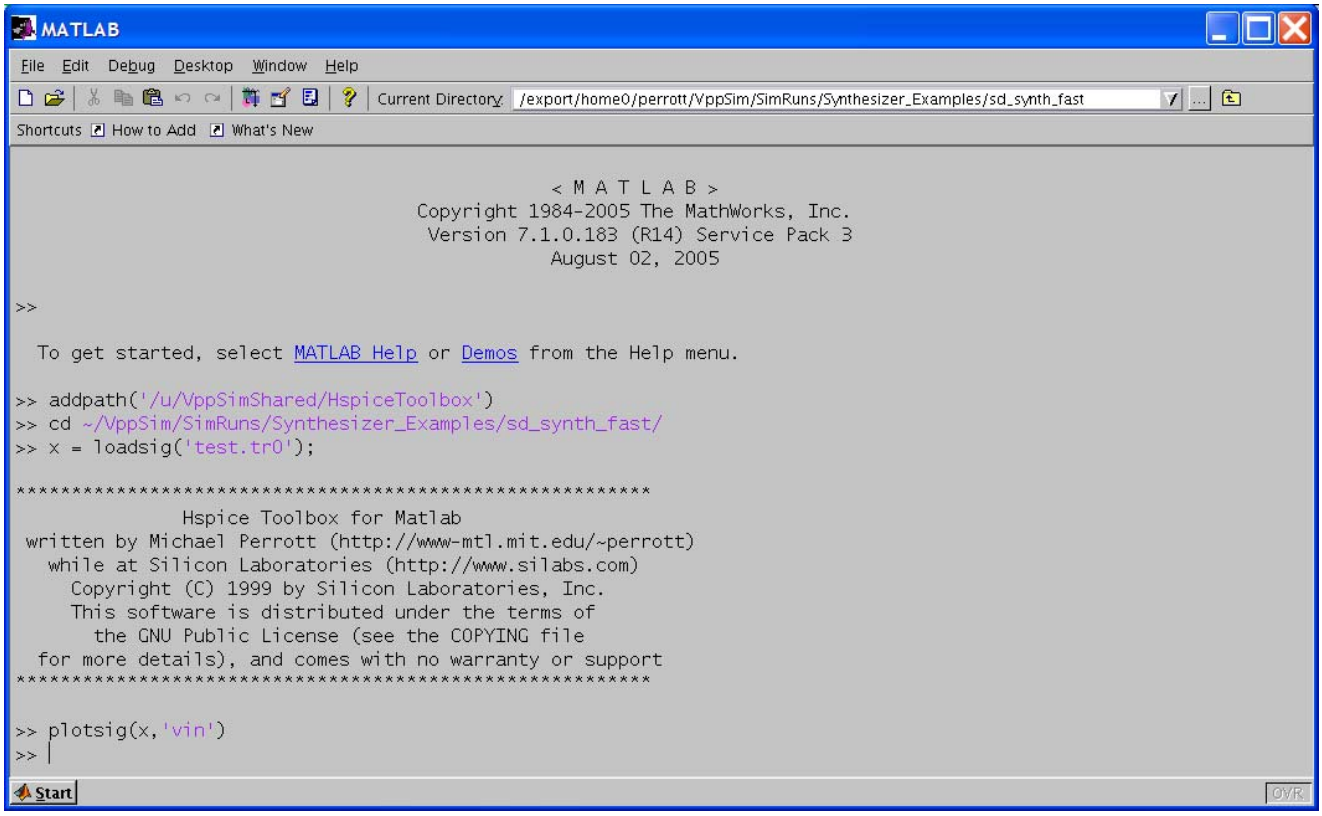
- Double-click on node **vin** to display a plot of this signal

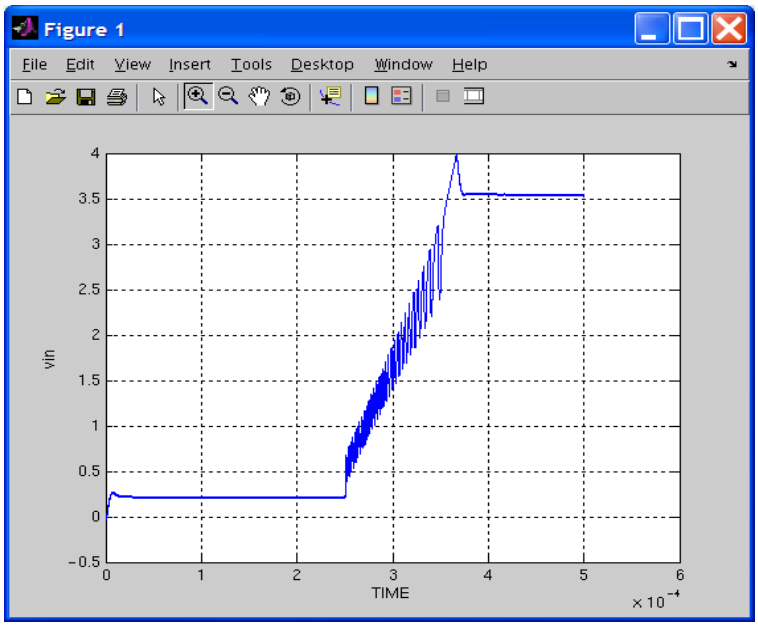




D. Viewing CppSim results within Matlab

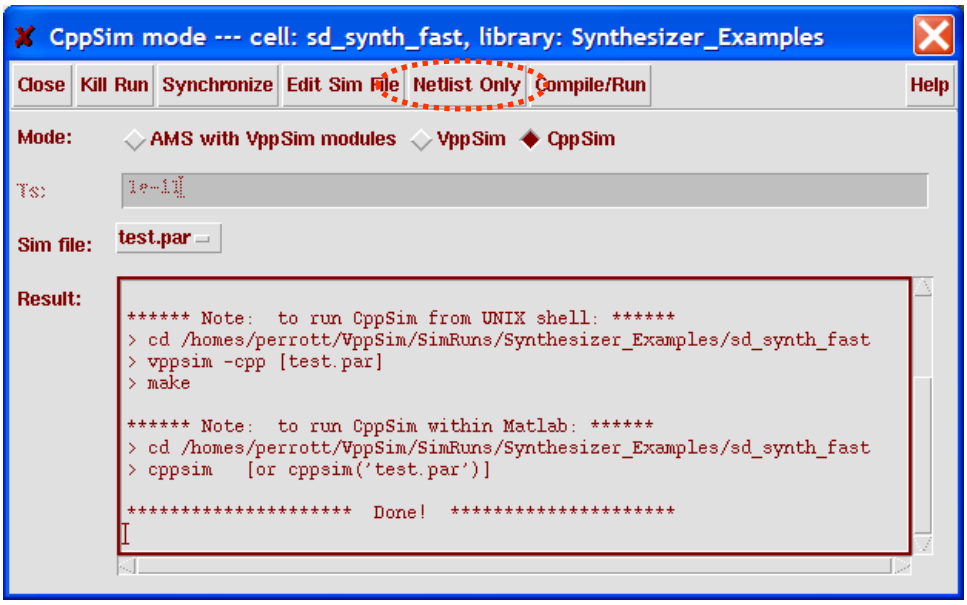
- Within Matlab, type the commands as shown below to see the associated plot of **vin**.



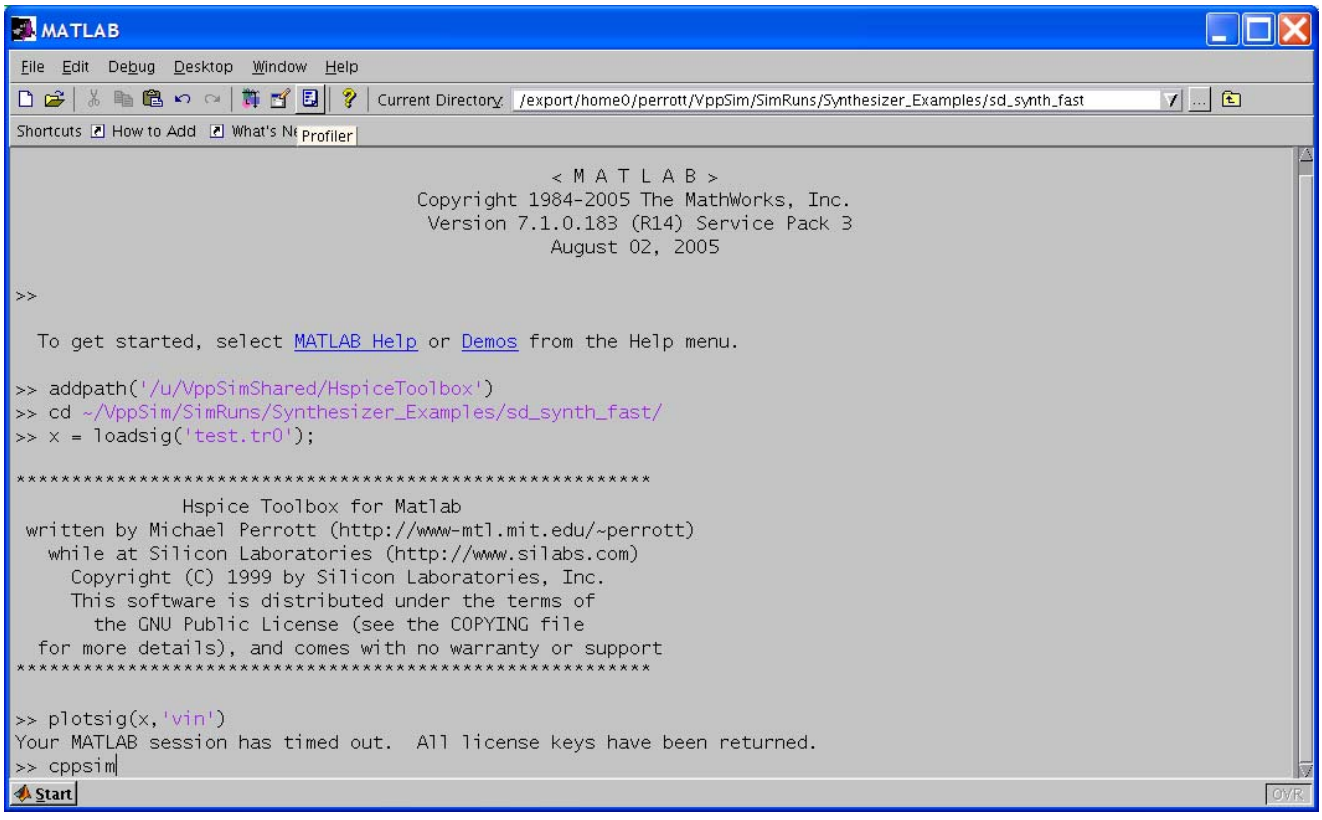


E. Running CppSim simulations within Matlab

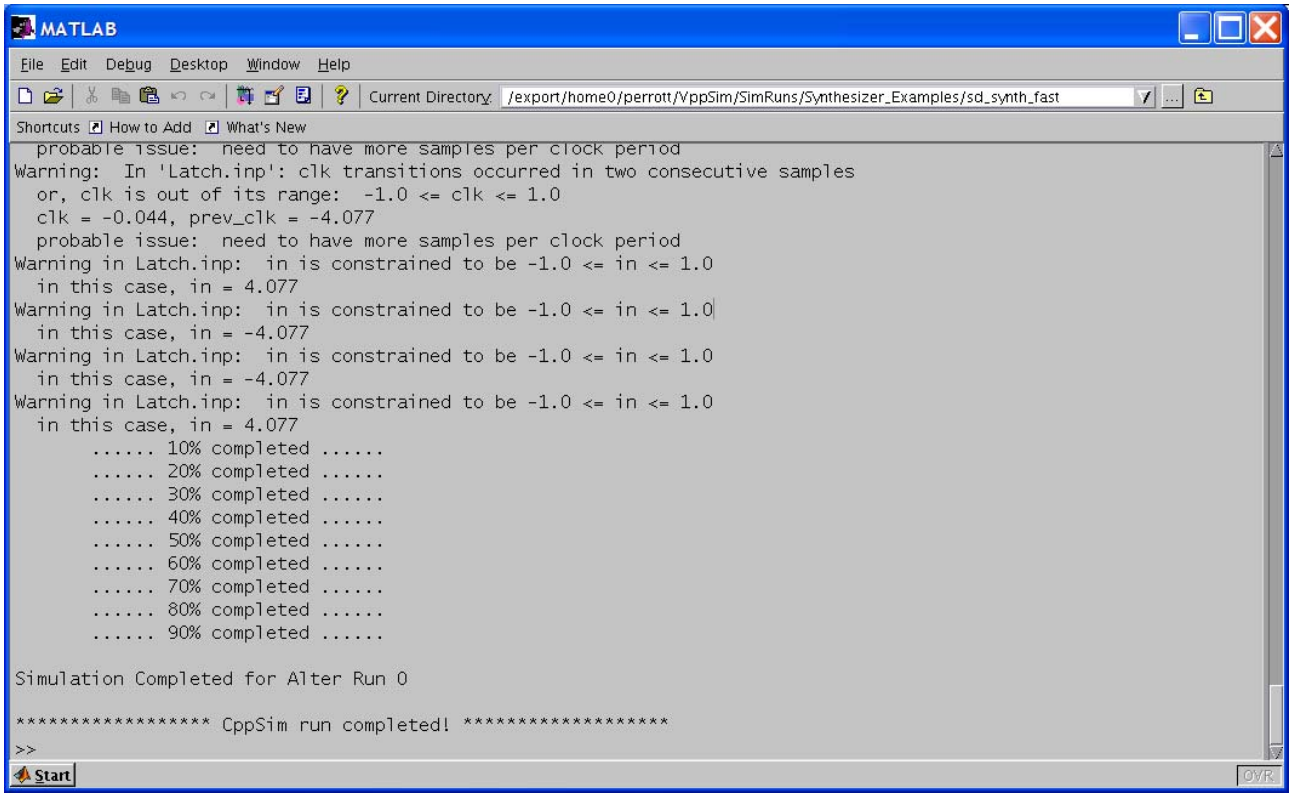
- Within the **CppSim/VppSim GUI** in Cadence, push the **Netlist Only** button. You will see instructions of how to run **CppSim** in both a Linux shell and in Matlab. Note that netlisting within the **CppSim/VppSim GUI** must always occur before changes to a schematic will be reflected in CppSim simulations run either from a Linux shell or Matlab.



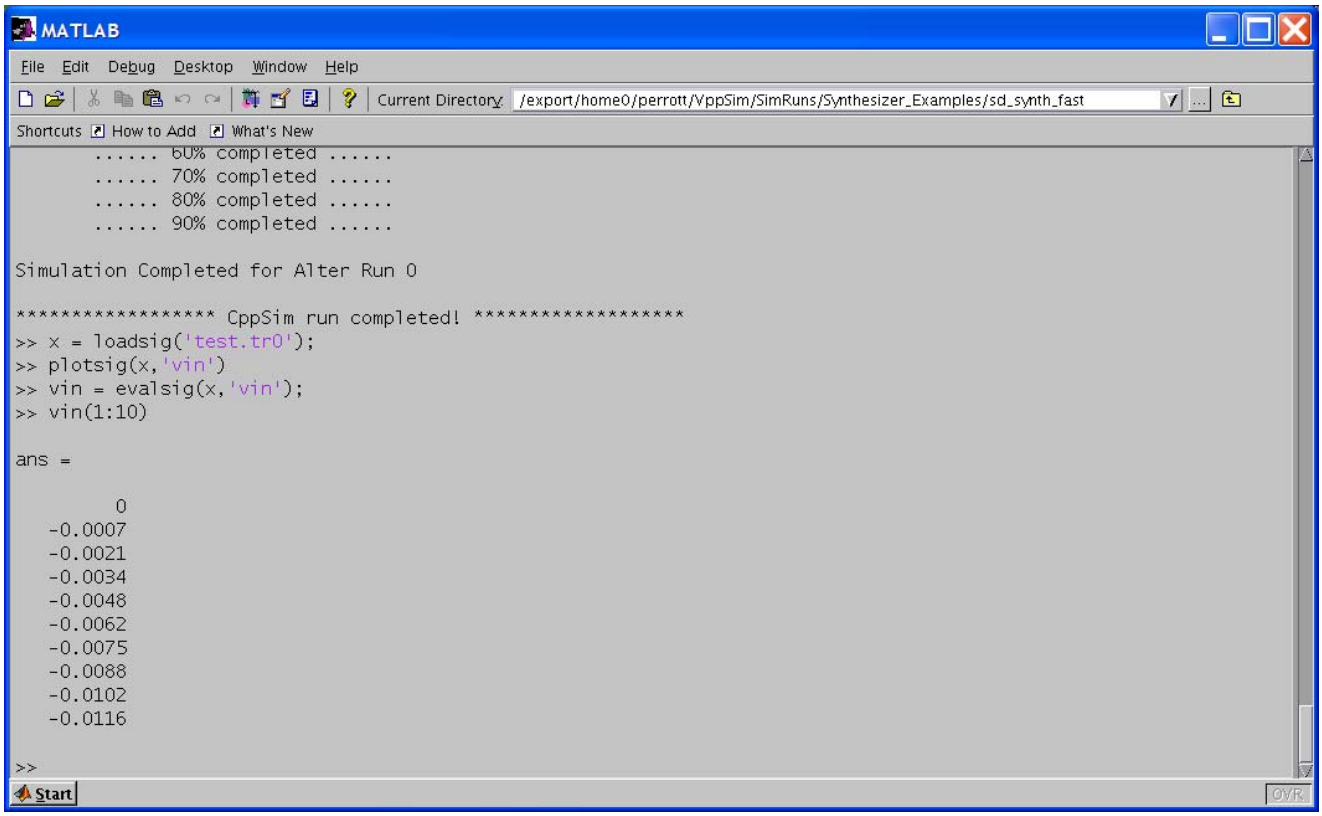
- Assuming that you have already run the Matlab commands from the previous section, simply execute **cppsim** at the Matlab prompt



- Upon completion of the CppSim run, the Matlab screen should appear as follows



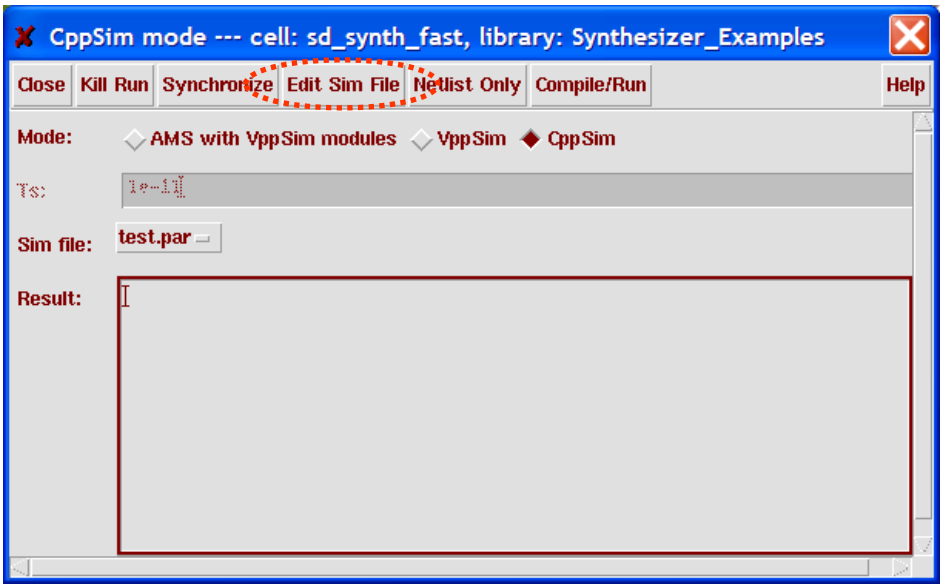
- Viewing or postprocessing of signals is easily achieved with the Hspice Toolbox commands, which are accessed by typing the following command in Matlab:
    - addpath('/u/CppSimShared/HspiceToolbox')



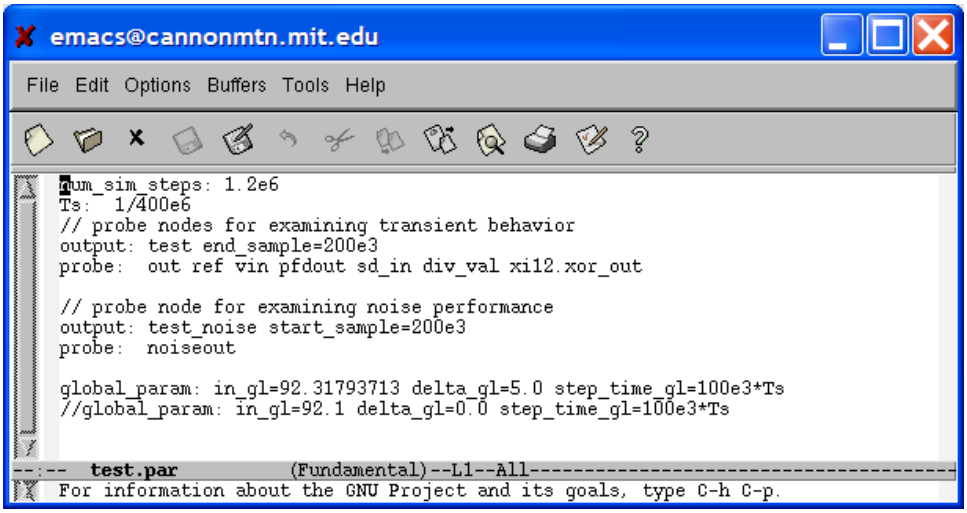
Setting CppSim simulation parameters for a given run

Simulation parameters for individual simulation runs are set in a file named test.par within your CppSim/SimRuns/ directory structure. The test.par file can be edited directly, or accessed using the CppSim/VppSim interface.

- Assuming that you have already opened the **sd_synth_fast** schematic view and its corresponding **CppSim/VppSim GUI** window (as described in the previous section), then push the **Edit Sim File** button within the **CppSim/VppSim GUI** window. In the case that there is no simulation file available, pushing of **Edit Sim File** creates a new one.



- A text Editor window should appear as shown below.
  - Notice that the number of time steps, the time step value, and the signals to be probed are all specified here.
  - To change any simulation parameters, modify them within the editor and then save the changes before your next run.



Examining existing CppSim modules

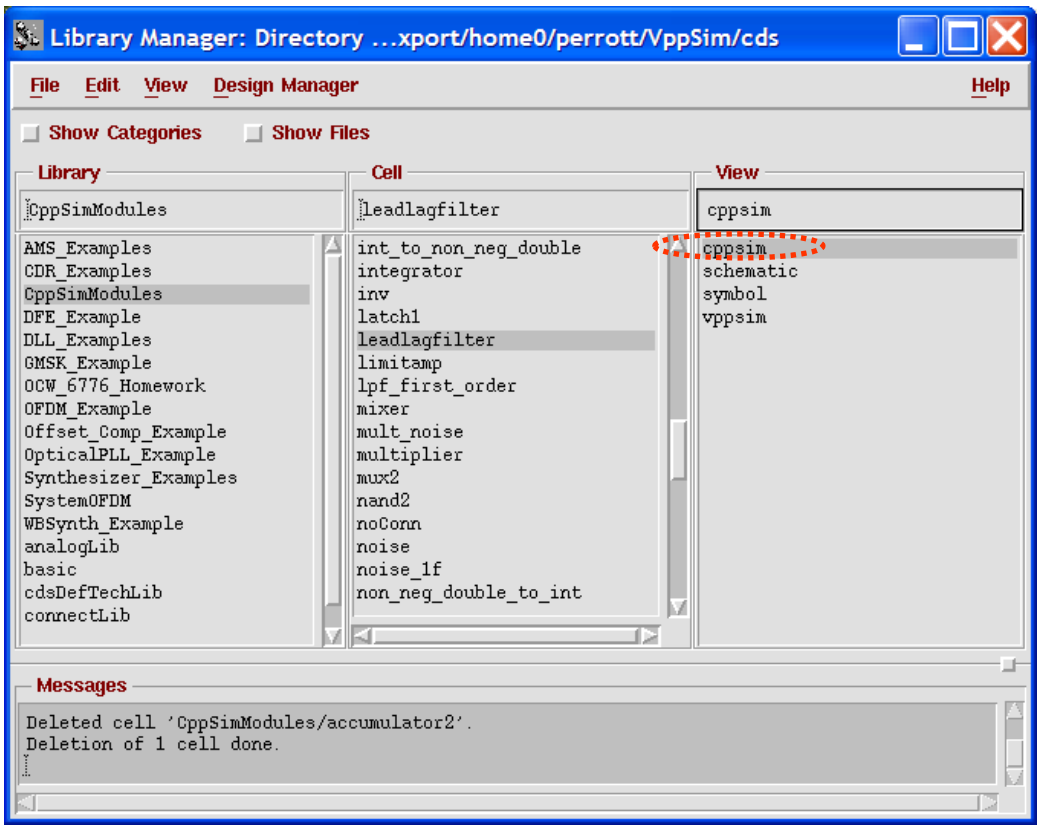
Within the CppSim framework, CppSim modules are placed within the Cadence design hierarchy as standard text files. Creation of their schematics, symbols, and parameters follows standard conventions for the Cadence package. Here we view the different components of an example cell that has already been created.

A. Starting Cadence

- Be sure to start Cadence within the **~/CppSim/cds** directory. This is necessary in order to access the appropriate **cds.lib** file and **.cdsinit file** available in that directory. Experienced Cadence users can move these files to a different directory of their choice, and start Cadence there.
  - `cd ~/CppSim/cds`
  - `icms & (or virtuoso & for Cadence 6)`

B. Examination of the cellviews of an existing CppSim module

- Within the **Cadence Library Manager**, double-click on the **cppsim** view of the **leadlagfilter** cell within the **CppSimModules** library.

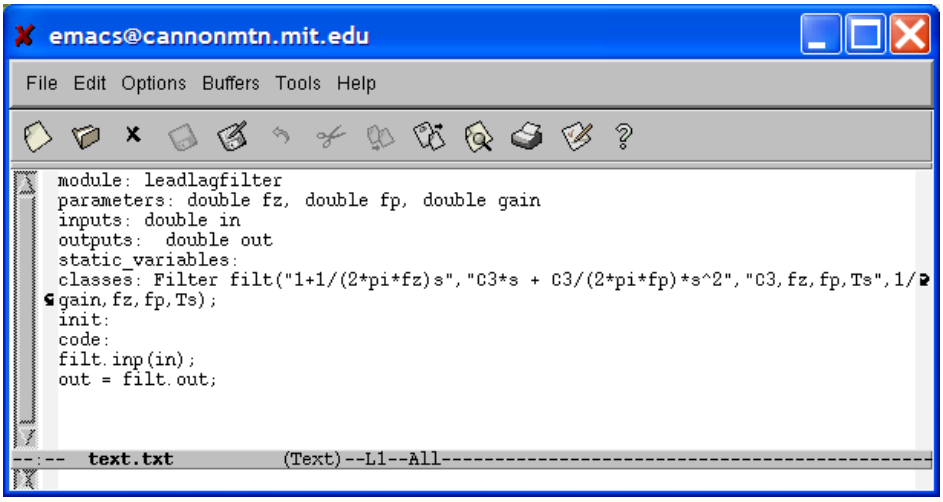


- An editor window should appear with the CppSim module code, as shown below.

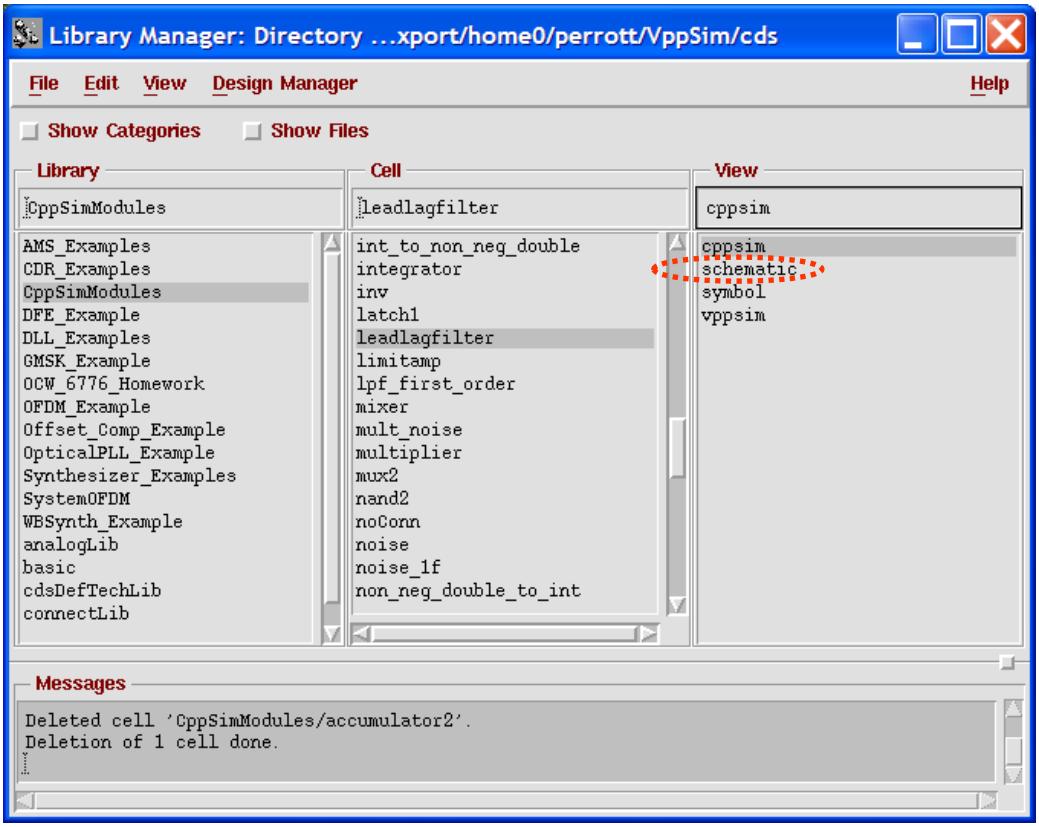
- Notice the template description of a CppSim module, which is described in fuller detail in the CppSim Reference Manual. The following sections comprise a CppSim module description file:

  - **module** – the name of the CppSim module
  - **description** (optional) – a high level description of the module
  - **parameters** – module parameters that can be set in the netlist editor
  - **inputs** – inputs to the module
  - **outputs** – the module's outputs
  - **classes** – a list of CppSim class instantiations the module will use
  - **static_variables** – variables that will preserve their state for the lifetime of the simulation
  - **init** – C++ code to initialize the static variables and outputs
  - **code** – C++ code that governs the behavior of the module
  - **end** (optional) – C++ code that will be executed at the end of the simulation

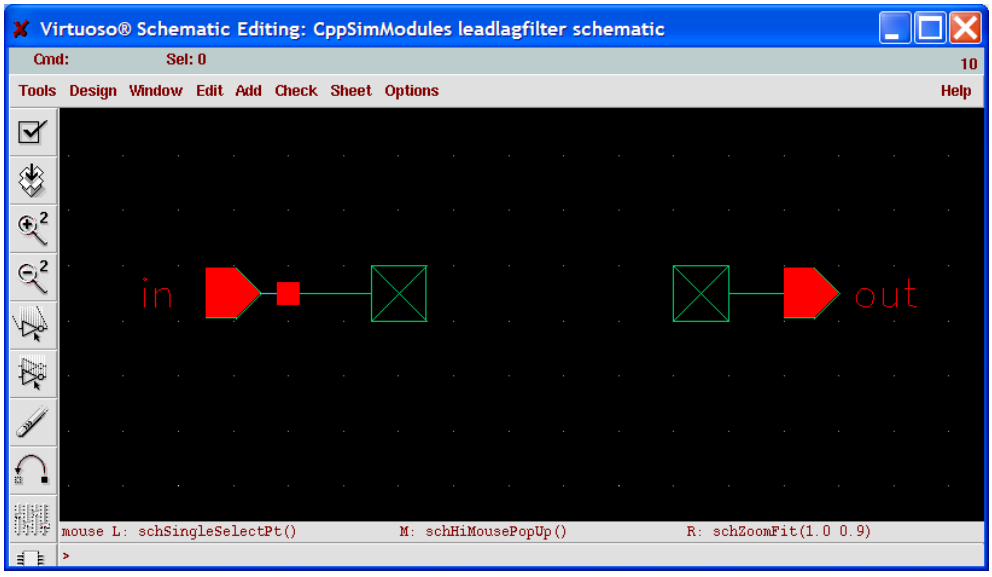
- You may close the editor once you have examined its contents.



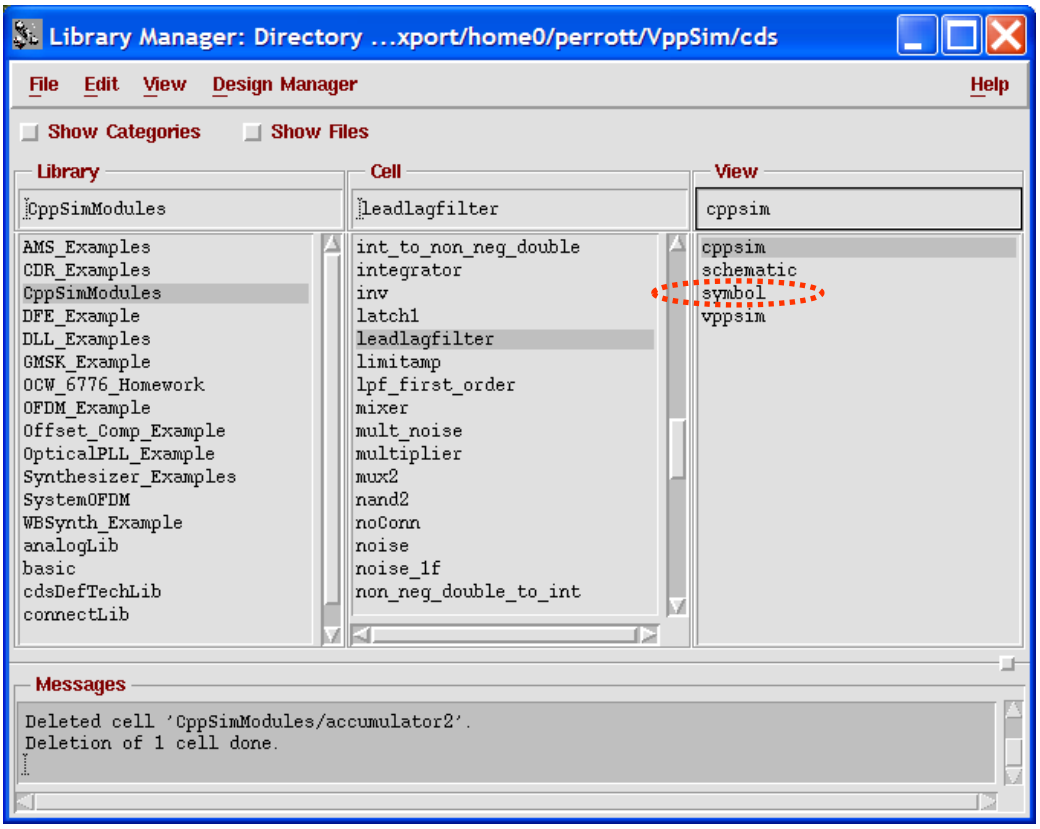
- Within the **Cadence Library Manager**, double-click on the **schematic** view of cellview **leadlagfilter** within the **CppSimModules** library.



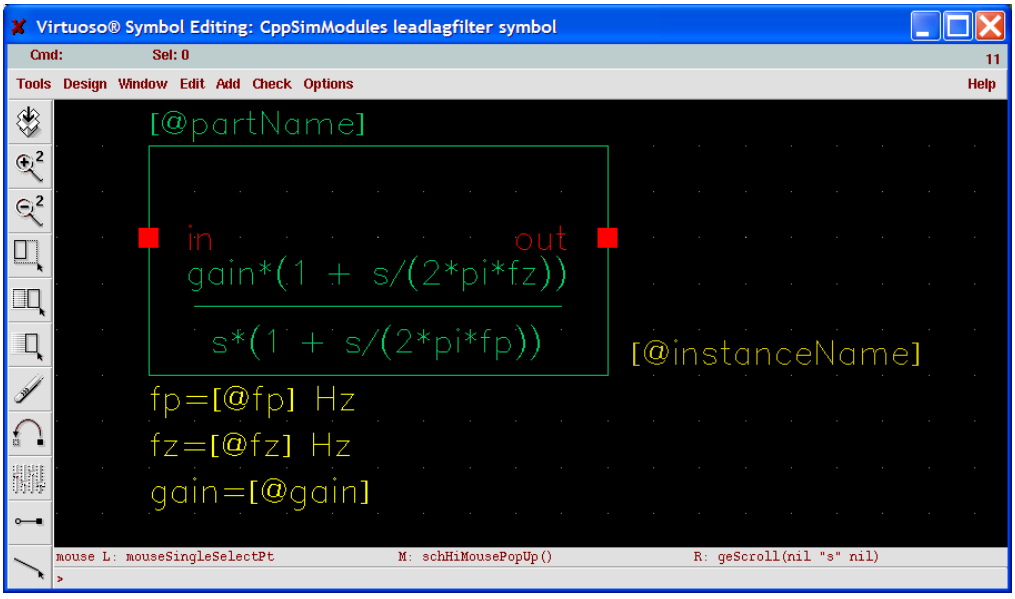
- A schematic view should appear as shown below.
    - Note this schematic only has input and output pins since it is represented by CppSim module code. However, it could also contain instances and circuit elements. In such case, CppSim can still represent the block as CppSim code, and thereby ignore the instances and circuit elements within the block. This feature allows one to represent an existing circuit with corresponding CppSim behavioral code without having to make any modifications to the circuit schematics.



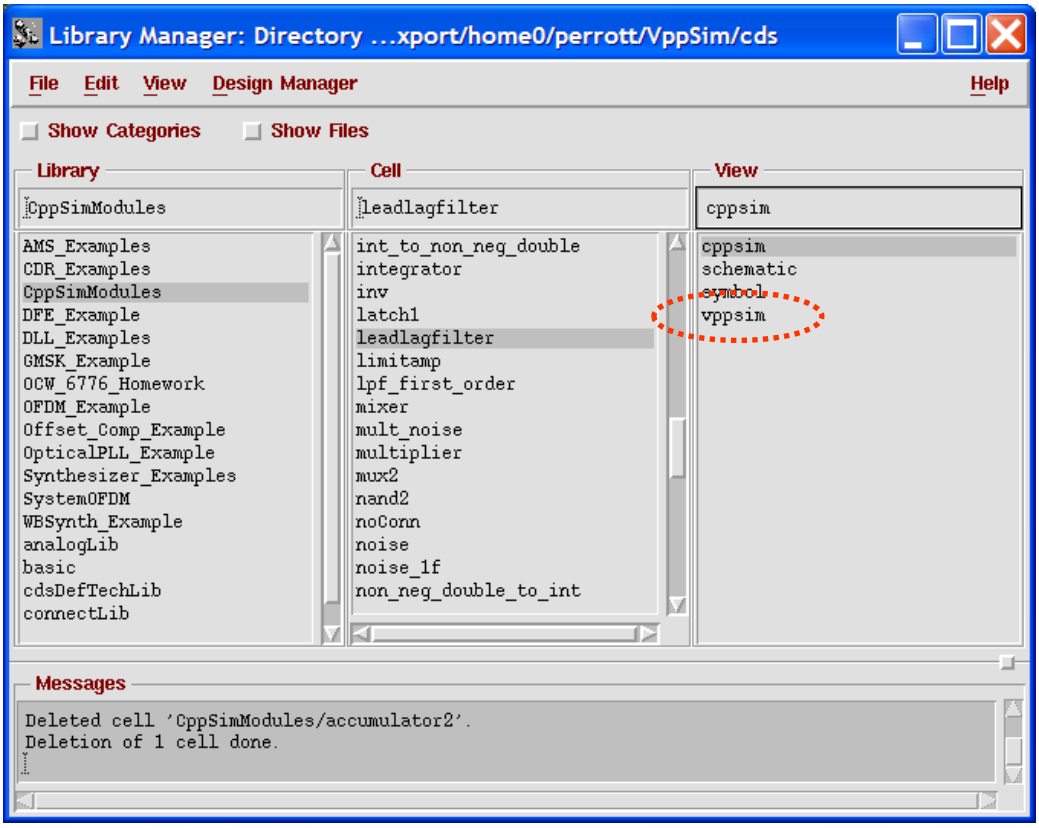
- Within the **Cadence Library Manager**, double-click on the **symbol** view of cellview **leadlagfilter** within the **CppSimModules** library.



- A symbol view should appear as shown below.
  - Notice that this symbol has three parameters associated with it: **fp**, **fz**, and **gain**. The statements shown here are used to display the value of the parameter. Creation of the parameters are achieved by using the **CDF** tool within the **Cadence CIW window,** and discussed shortly.
  - Note also the following conventions in creating CppSim symbols:
    1. Place pins and external shapes on grid.
    2. Space the pins at least one grid away from the edge of the shape and other pins
    3. Pins should be square with no legs (logic gates, etc. can be exceptions)
    4. Shape outline should conform to visible shape (not outside text, etc.)
    5. Justify all parameters, partnames, and instancenames to centerLeft
    6. [@partName] should be placed over the upper left of the shape, justified with the left edge
    7. [@instanceName] should be to the right of the bottom right shape corner
    8. parameters should be below the bottom left shape corner, with multiple parameters spaced one grid apart
    9. All text size should be 0.0625 in stick font (default)
    10. Write units whenever possible (i.e., Hz) for parameters



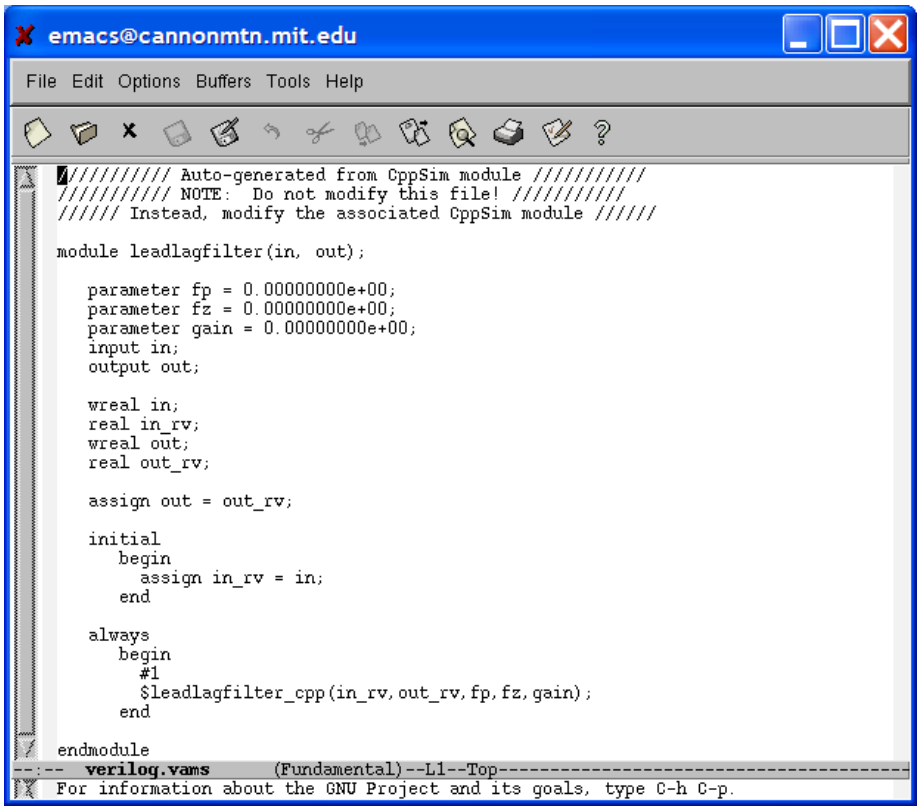
- Within the **Cadence Library Manager**, double-click on the **vppsim** view of cellview **leadlagfilter** within the **CppSimModules** library.



- An editor window should appear with the VppSim module code, as shown below.
  - Note that this code was previously auto-generated using the **AMS with VppSim Modules** option within the **CppSim/VppSim GUI** window.
  - Note that the **vppsim** code corresponds to a Verilog-AMS module, and that it consists of a Verilog-AMS wrapper that calls the PLI function **$leadlagfilter(…)**.

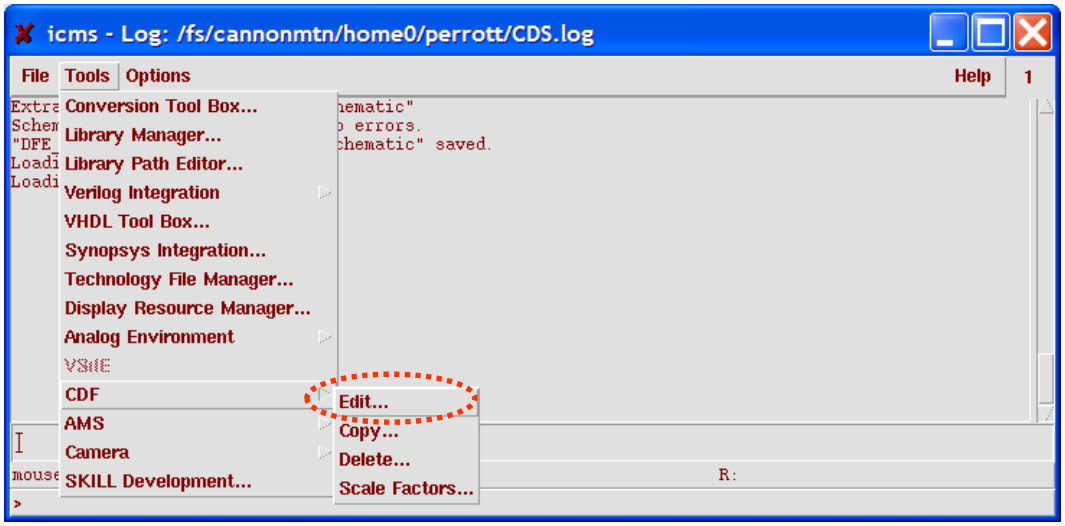
This PLI function is automatically created when using the **AMS with VppSim Modules** option within the **CppSim/VppSim GUI** window, and included by using the **loadpli1** option within the **Elaborator Options** interface of AMS.

- More details of including VppSim modules within AMS are discussed later in this document.

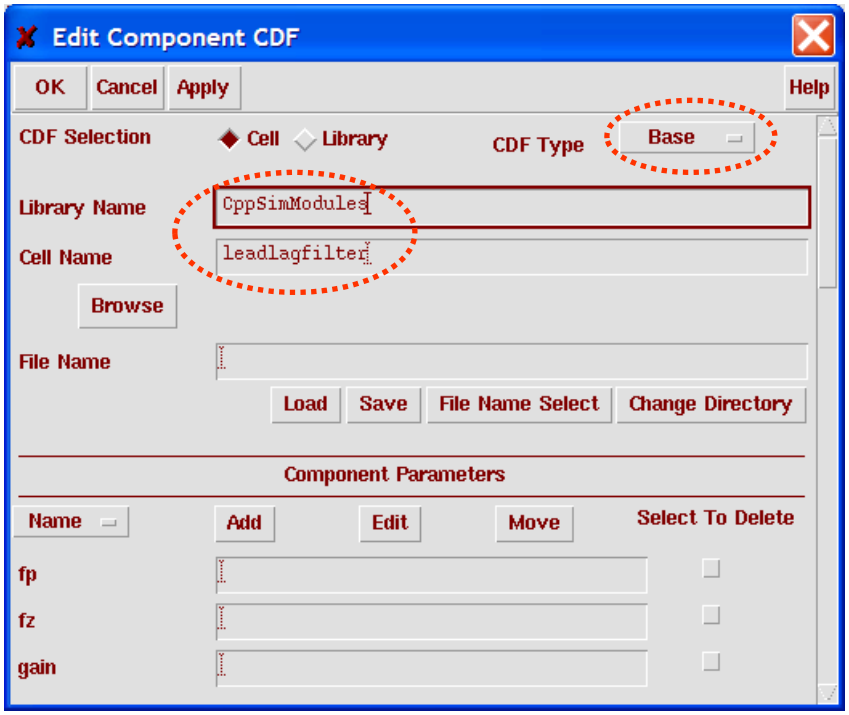


C. Examination of the CDF parameters of an existing CppSim module

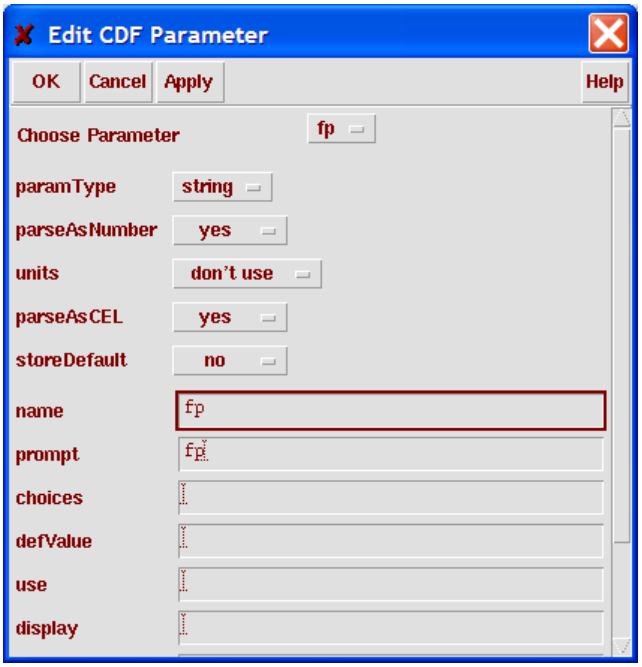
- Within the **Cadence CIW window**, click on **Tools->CDF->Edit**.



- Within the **Edit Component CDF** window that pops up, select the **CDF Type** to be **Base**, and the **Library** and **Cell** names to be **CppSimModules** and **leadlagfilter**, respectively.



- Click on **Edit** within the **Component Parameters** section of the above window. An **Edit CDF parameter** window should appear as shown below.
    - Take careful note of the options shown in the form below (i.e., **paramType**, **parseAsNumber**, etc.). *These same option settings must be used for ALL CppSim modules.*



Creating new CppSim modules

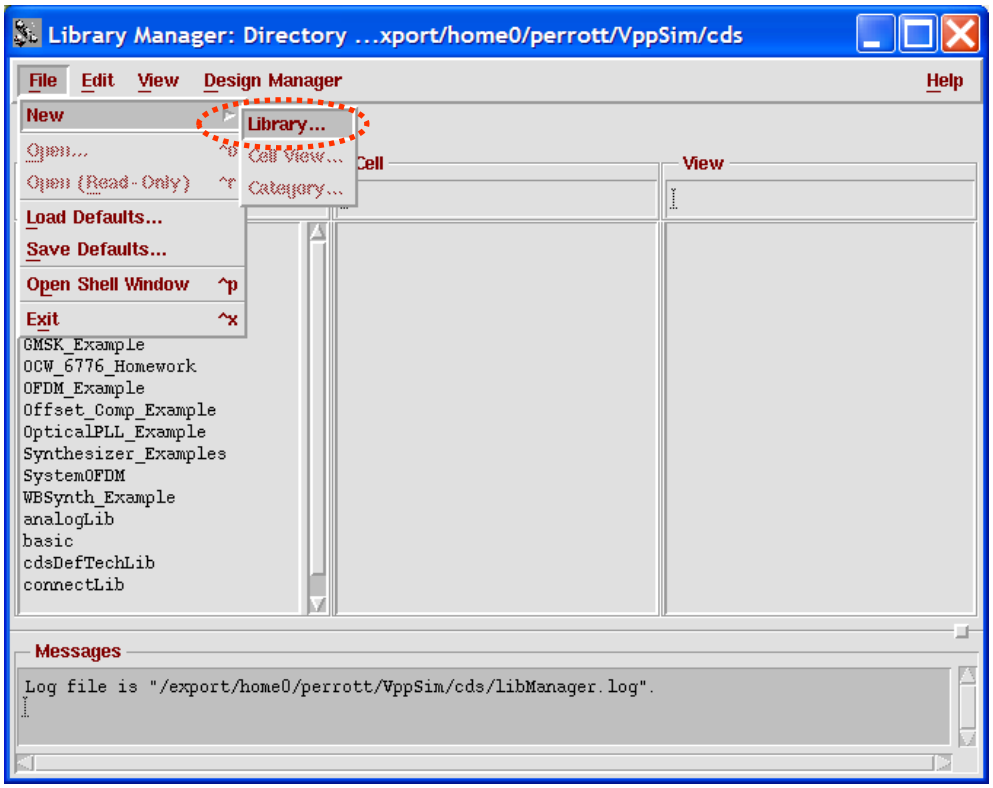
This section will describe how to create new CppSim modules. It is assumed that you have already read the previous section which describes the basic structure of CppSim schematics, symbols, parameters, and code.

A. Starting Cadence

- Be sure to start Cadence within the **~/CppSim/cds** directory. This is necessary in order to access the appropriate **cds.lib** file and **.cdsinit file** available in that directory. Experienced Cadence users can move these files to a different directory of their choice, and start Cadence there.
  - `cd ~/CppSim/cds`
  - `icms & (or virtuoso & for Cadence 6)`

B. Creation of new Cadence library

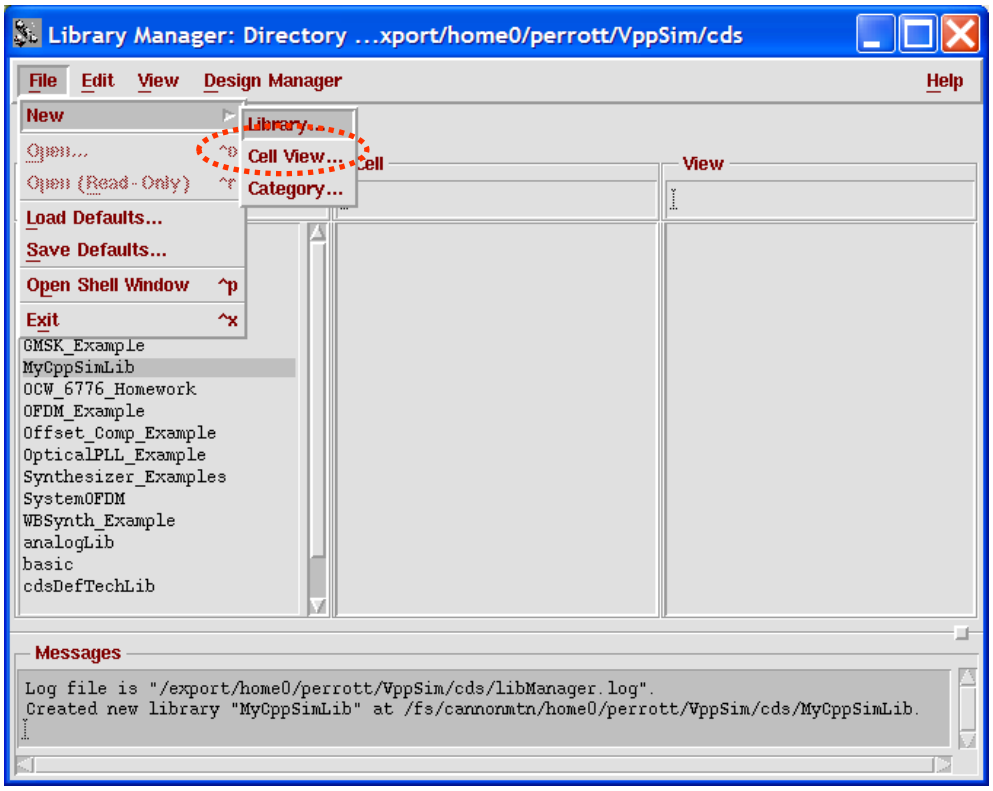
- Select **File->New->Library** within the **Cadence Library Manager**.



- In the window that appears, name the library **MyCppSimLib**. After pressing **OK**, designate that you do not need a techfile within the form that appears.

C. Creation of a new CppSim module schematic

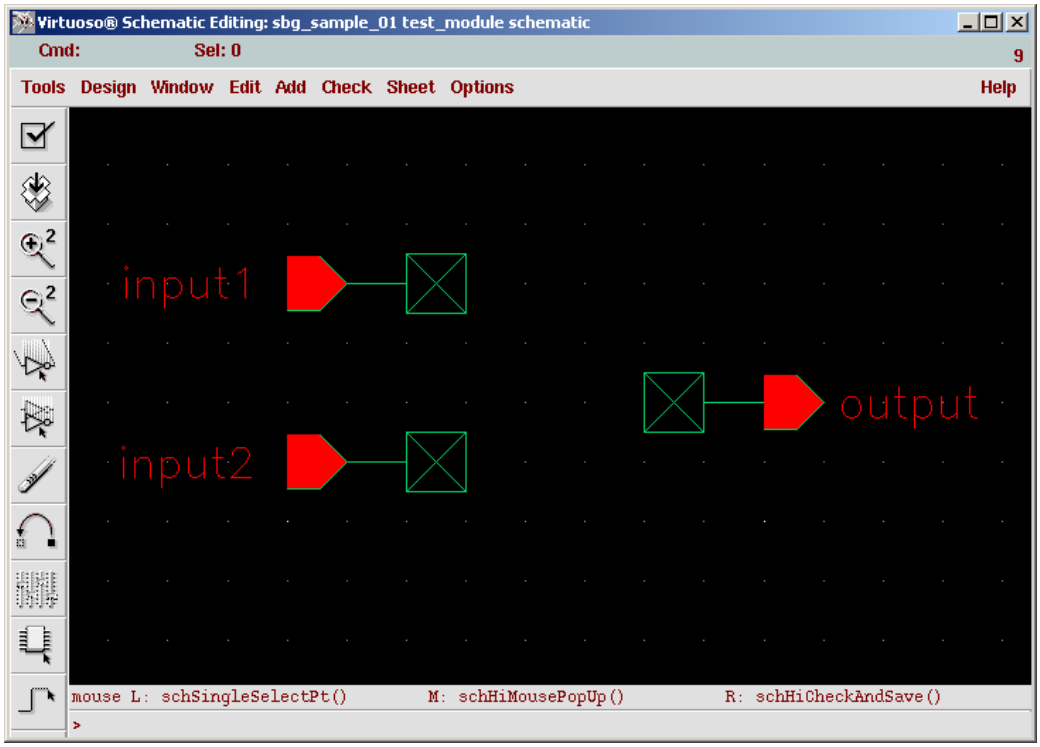
- Select **MyCppSimLib** within the **Cadence Library Manager** and then select **File->New->Cell View**.
- In the window that appears, choose the **Cell Name** to be **my_cppsim_module** and then press **OK**.



- Within the newly opened schematic window, create 2 input and 1 output pins named **input1, input2** and **output**, respectively. Attach **noConn** instances from the **CppSimModules** library to each of the pins in order to avoid warning messages when you save the schematic. Here is a list of Cadence command shortcuts that may be helpful to you:

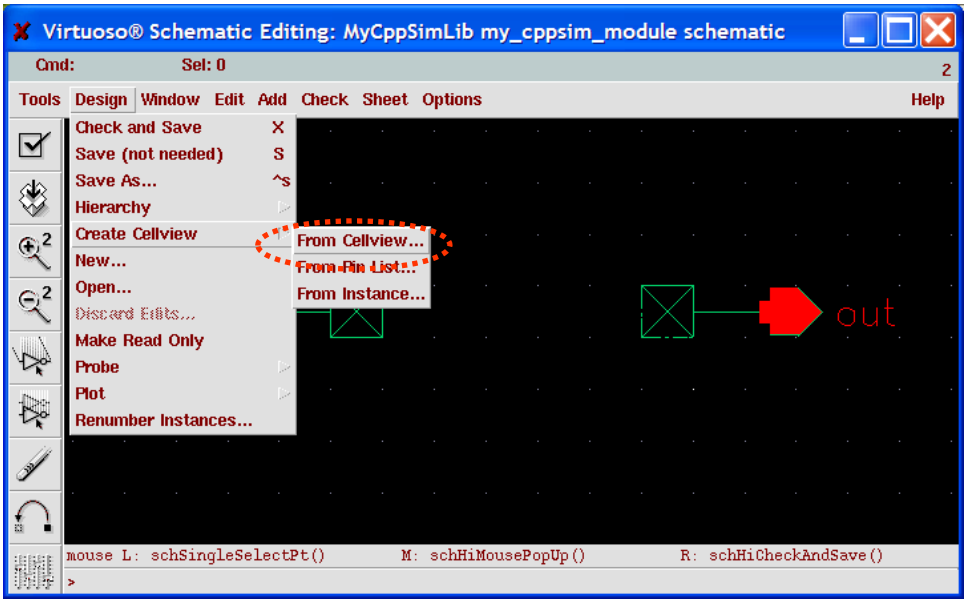
    - **i** – add instance of module
    - **p** – add input/output pin
    - **w** – add wire
    - **l** – add label
    - **q** – view/edit object parameters
    - **r** – rotate object
    - **Delete** – delete object
    - **u** – undo
    - **f** – zoom fit
    - **[** – zoom out
    - **]** – zoom in
    - **Arrow keys** – pan up/down/left/right
    - **S** – save
    - **F7** – close window

- Press the check-and-save button to save the schematic. Upon completion, your schematic should appear as below.



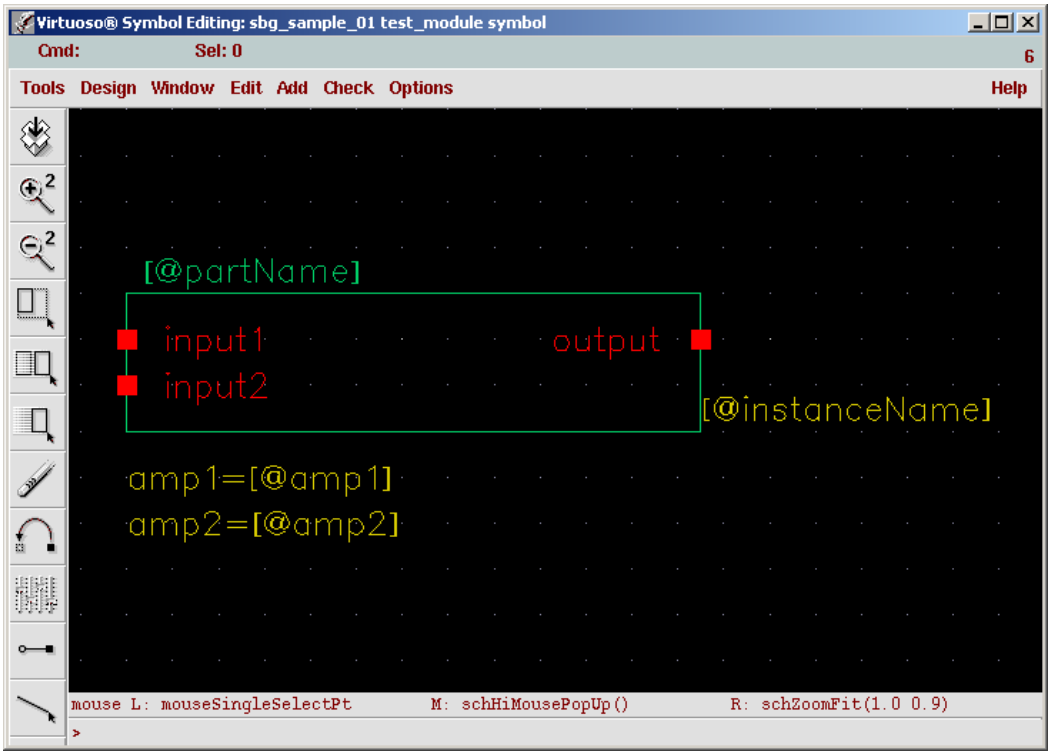
D. Creation of a new CppSim module symbol

- Within the schematic window, select **Design->Create Cellview->From Cellview** as shown below. Press **OK** on the forms that follow.



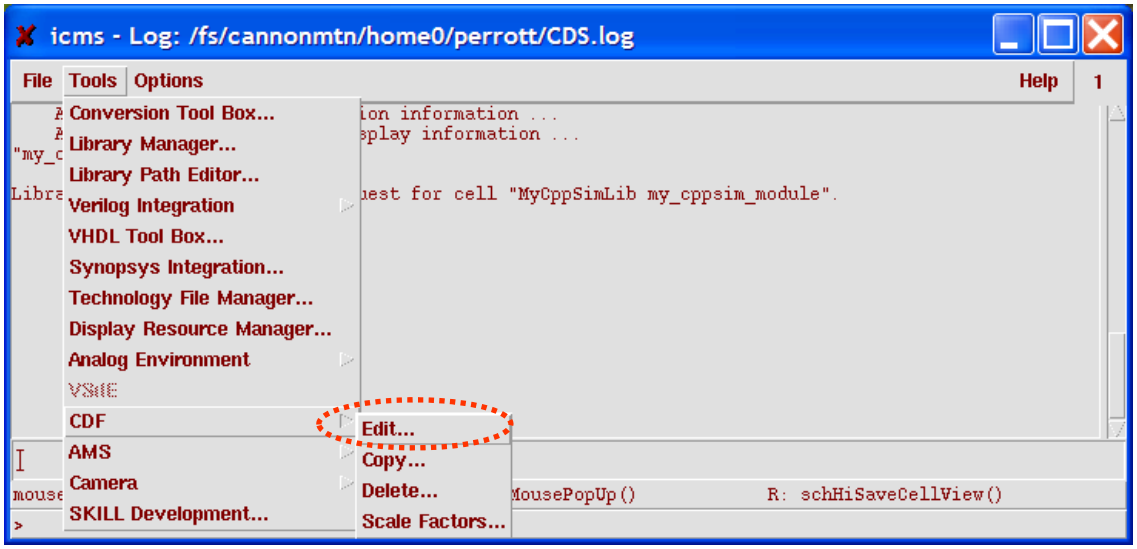
- A default symbol window should appear. We will make a few cosmetic changes, and then add text for parameters.

- As discussed in part B of the previous section, it is nice to follow certain cosmetic rules when creating symbols. Following those rules, we adjust the symbol drawing and add parameter text as shown below. Be sure to save the symbol once you have completed your changes.



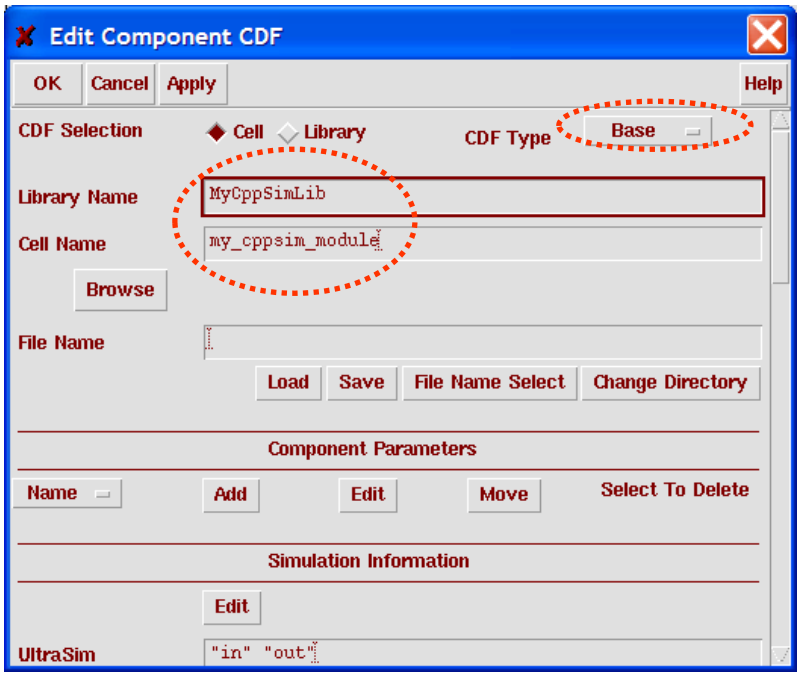
E. Creation of new CppSim module parameters

- Within the **Cadence CIW** window, select **Tools->CDF->Edit**.

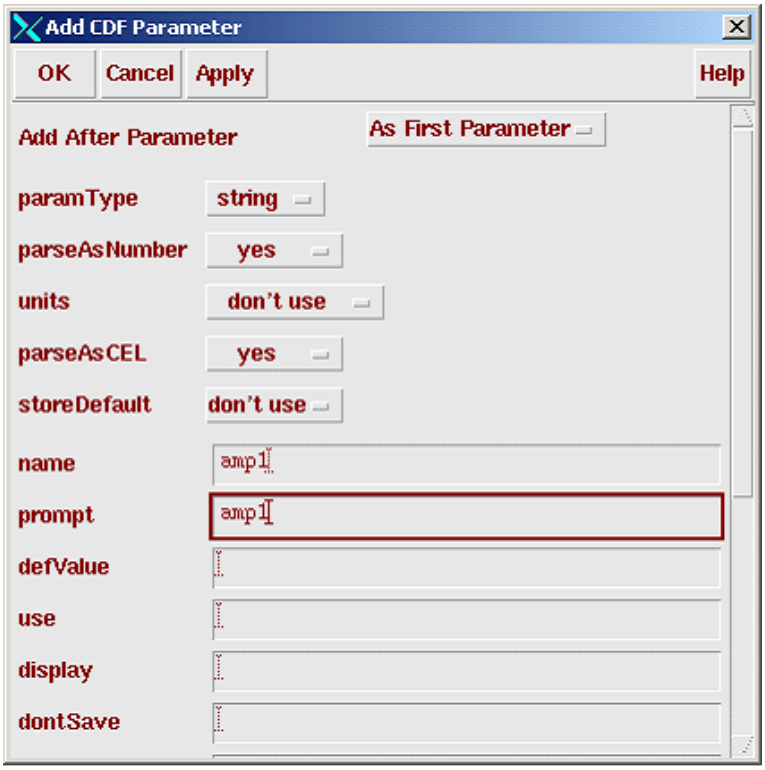


- In the **Edit Component CDF** window that appears, choose the **CDF Type** to be **Base**, and then enter **MyCppSimLib** and **my_cppsim_module** as the **Library** and **Cell** names,

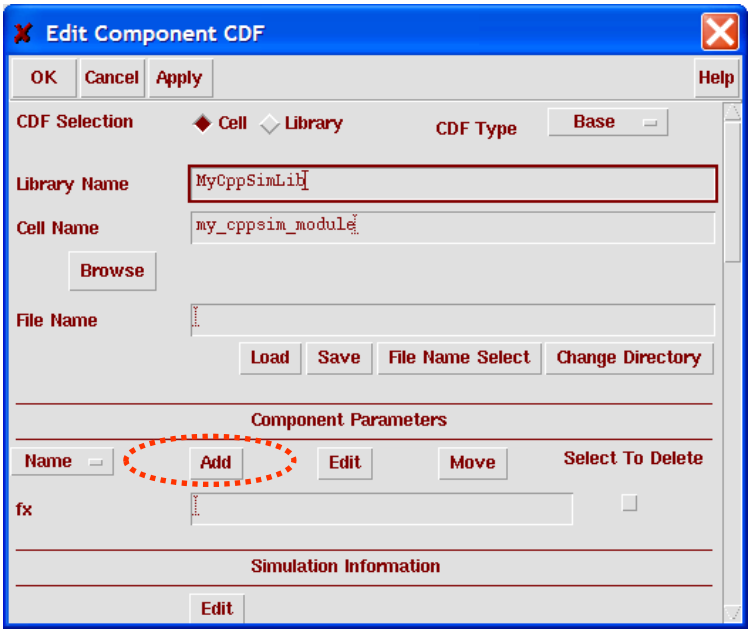
respectively. (NOTE: you may also click “Browse” and locate the module graphically rather than typing in the name)



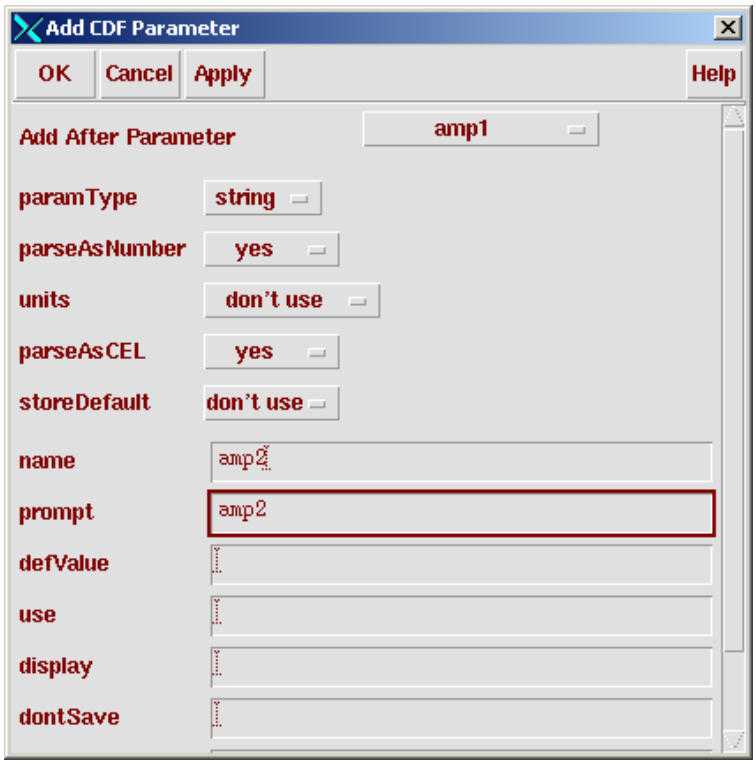
- Click on the **Add** button within the **Component Parameters** section of the above window. Within the **Add CDF Parameter** window that appears, make the selections and form entries as shown below. Press **OK** when you have completed the changes. (NOTE: remember to *always* use these settings for `paramType`, `parseAsNumber`, `units`, `parseAsCEL`, and `storeDefault`)



- Within the **Edit Component CDF** window, click on **Add** again within the **Component Parameters** section.

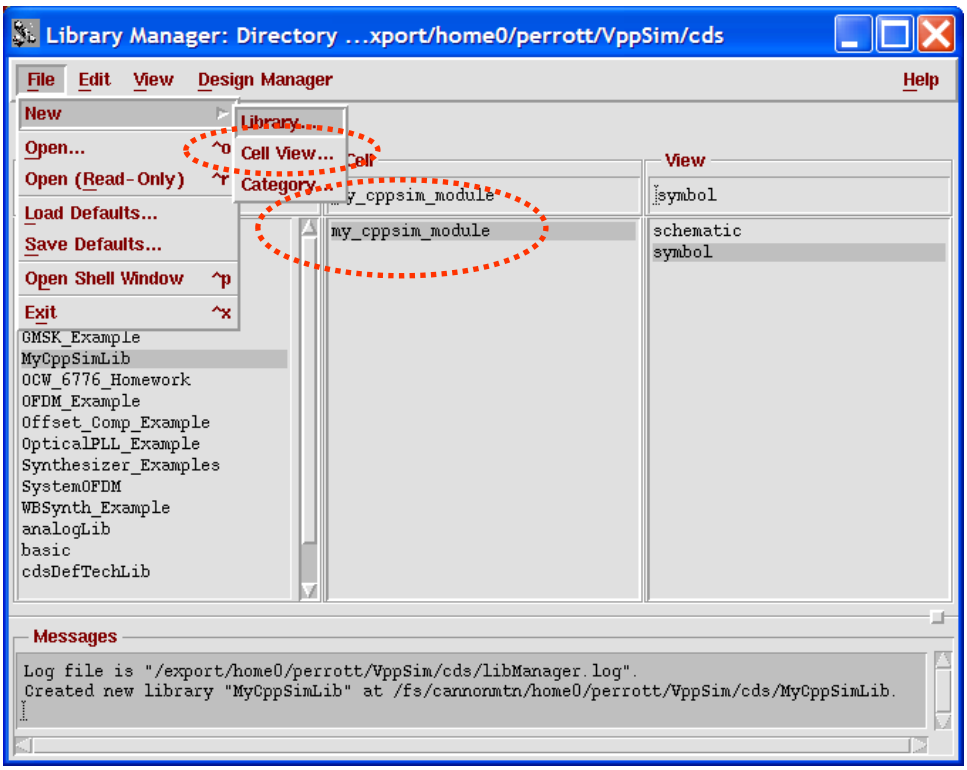


- Within the **Add CDF Parameter** window that appears, make the changes as indicated below. In particular, be sure to choose the parameter to be added after parameter **amp1**. This eases the editing of parameters in schematics by having the order of parameters listed in the symbol match that of the parameter edit form. Press **OK** when you have completed the changes below, and then **OK** in the **Edit Component CDF** window.

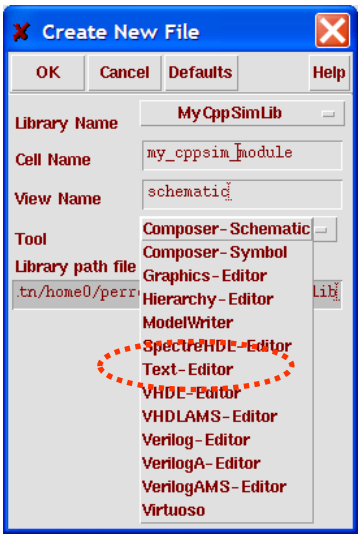


F. Creation of new CppSim module code

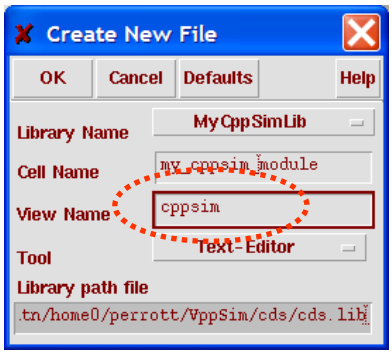
- Within the **Cadence Library Manager**, select **MyCppSimLib** and then **File->New->Cell View** as shown below.



- Within the Create New File window that opens, select Text-Editor as the Tool.



- Designate the **View Name** to be **cppsim**, and then push **OK**. (NOTE: this is where it is important to have your EDITOR environment variable set to a graphical editor)



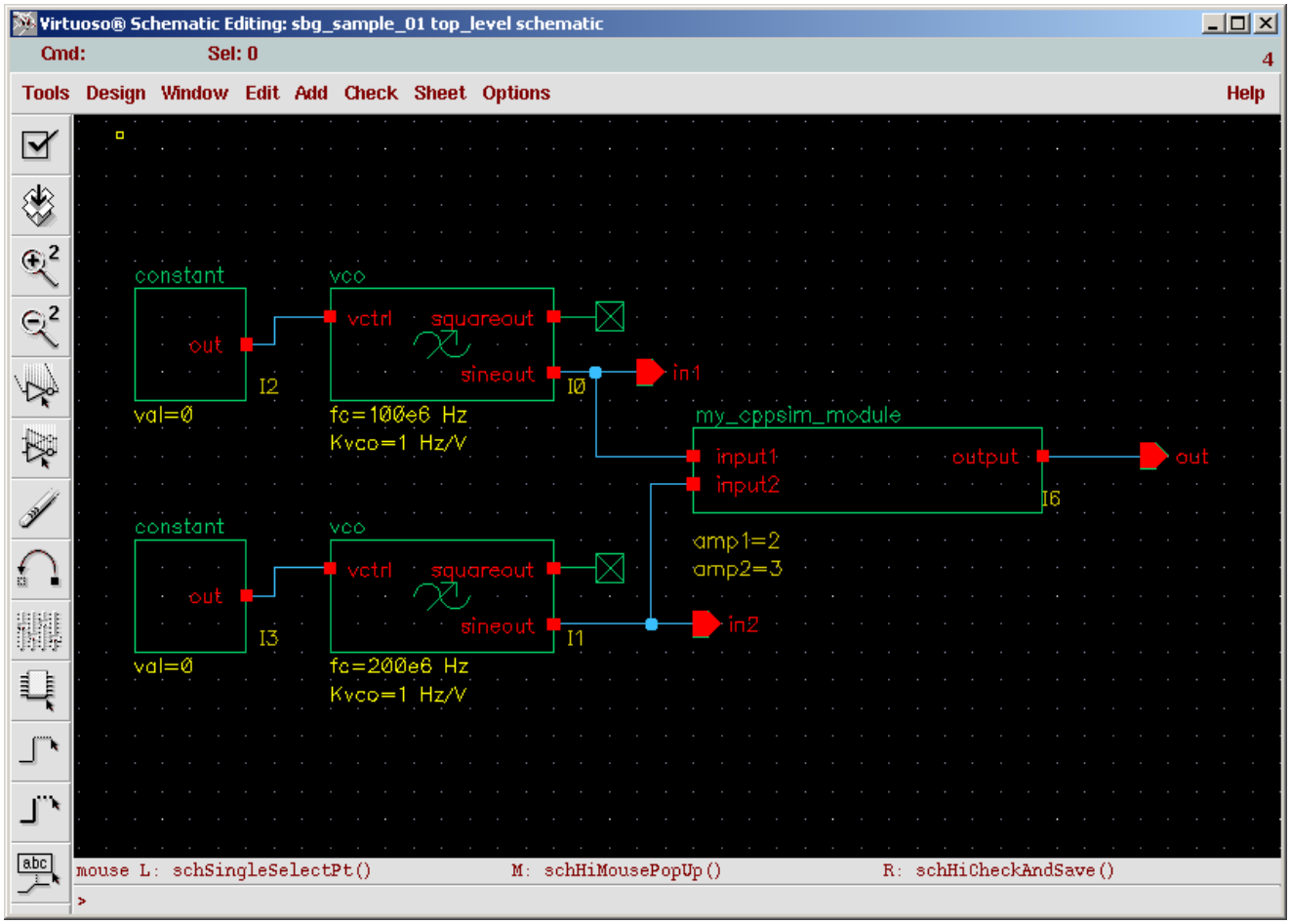
- A blank Editor window should appear, within which you should fill in the CppSim module description. Enter the following description:

```
module: my_cppsim_module

description:

parameters:
    double amp1;
    double amp2;

inputs:
    double input1;
    double input2;

outputs:
    double output;

classes:

static_variables:

init:

code:
    output = amp1 * input1 + amp2 * input2;

end:
```

G: Testing your new CppSim module

- From the **Cadence Library Manager**, create a new schematic Cell View named **top_level**

- Build the following structure in the top_level schematic and use it to test the module you created using the procedures described in previous sections: (NOTE: all modules that are not a part of your library can be found in the **CppSimModules** library)



.

VppSim Simulations

VppSim provides translation of a CppSim simulation to Verilog code combined with PLI modules so that ncverilog (or Icarus Verilog) can be used as the simulator. One can get a sense of how it works by examining a tutorial for MIT class 6.973 (taught by Vladimir Stojanovic) which is available at http://www.cppsim.com. Documentation for the VppSim tool will be expanded in the future.

Incorporating VppSim modules within AMS

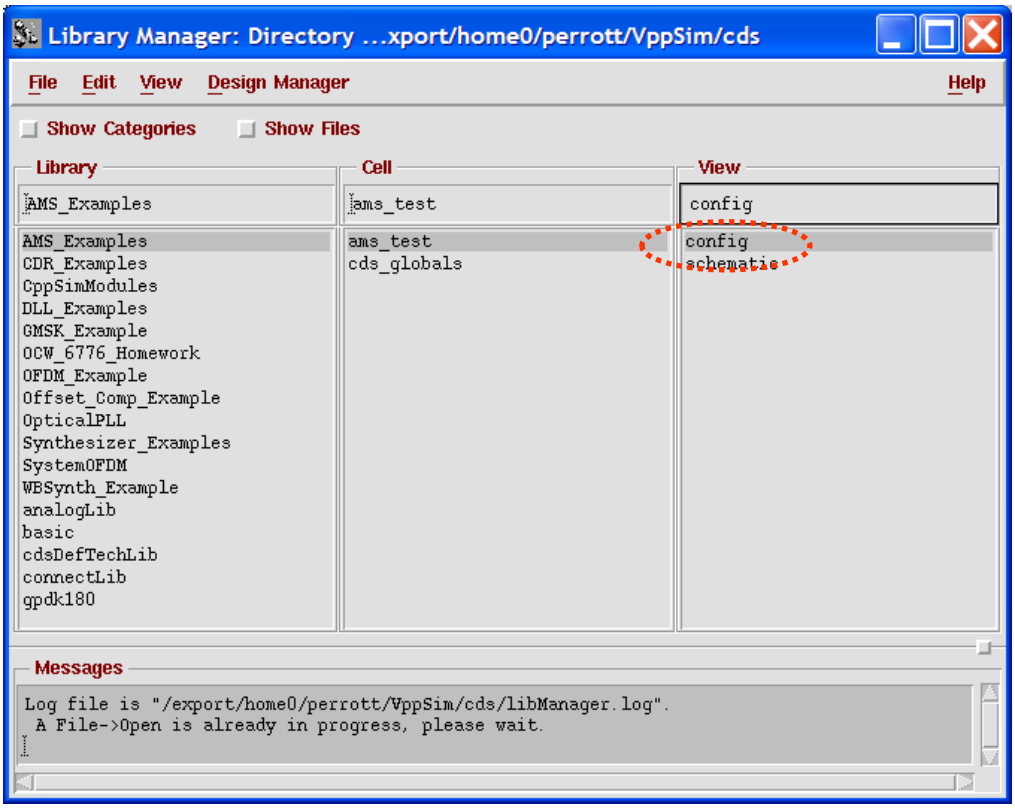
CppSim modules can also be used within AMS, so that seamless co-simulation can be achieved with SPICE, Verilog, and VHDL modules. To do so, the CppSim modules are embedded within Verilog-AMS wrapper code that allows them to be treated as standard Verilog-AMS modules. The auto-generated Verilog-AMS modules are referred to as VppSim modules. It is important to realize that VppSim modules directly execute the C++ code corresponding to their respective CppSim module – no translation is done. Rather, the C++ code is simply wrapped within a Verilog module to allow seamless incorporation within the Verilog and AMS simulation environments.

A. Starting Cadence

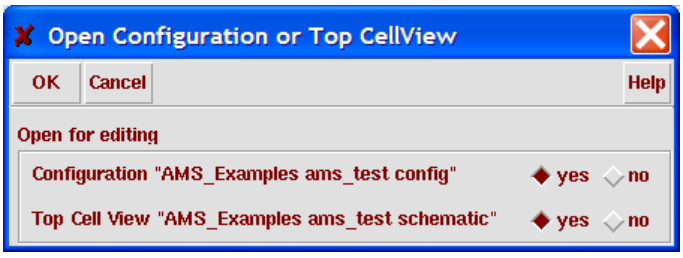
- Be sure to start Cadence within the **~/CppSim/cds** directory. This is necessary in order to access the appropriate **cds.lib** file and **.cdsinit file** available in that directory. Experienced Cadence users can move these files to a different directory of their choice, and start Cadence there.
    - In Linux shell window:
      > cd ~/CppSim/cds
      > icms & (or virtuoso & for Cadence 6)

B. Opening AMS Example using VppSim modules

- Double-click on the **config** view of the **ams_test** cellview within the **AMS_Examples** library in the **Cadence Library Manager**.



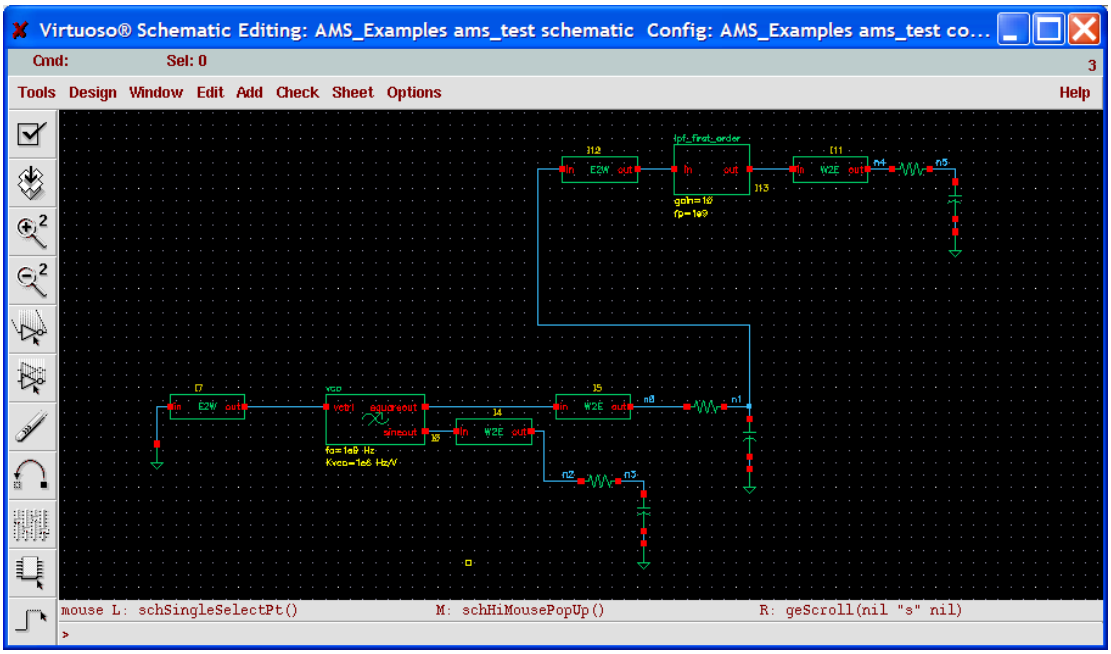
- Click the **yes** radio buttons on the resulting dialog box.



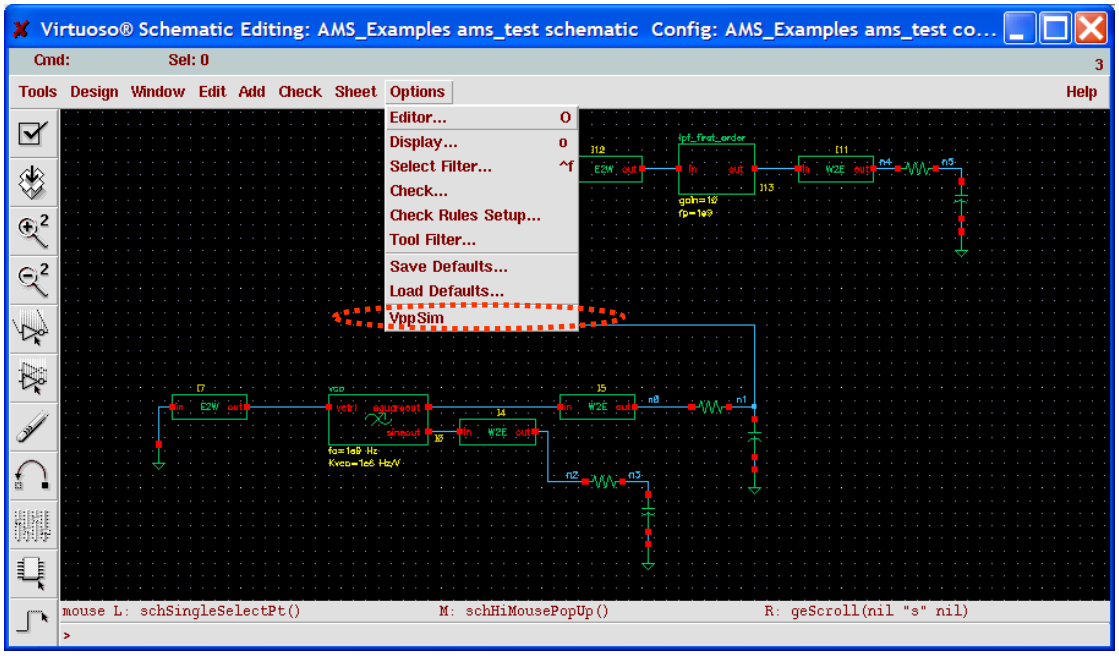
- Within the resulting schematic window, take note of the following items:

- **W2E** modules: translate from wreal signals (i.e., double-valued signals within the Verilog-AMS environment) to electrical signals (i.e., SPICE signals).
- **E2W** modules: translate from electrical signals (i.e., SPICE signals) to wreal signals (i.e., double-valued signals within the Verilog-AMS environment).
- **vco** and **lpf_first_order** modules: CppSim modules that are represented by their **vppsim** view within the AMS simulation.
- RC networks: examples of SPICE-level components.

Note that the **W2E** and **E2W** modules could be automatically inserted by AMS through proper set up of the AMS configuration. Here we show these translation modules for simplicity in illustrating the key ideas of the example.



- Click on **Options->CppSim/VppSim** within the resulting schematic window in order to bring up the **CppSim/VppSim GUI** window.

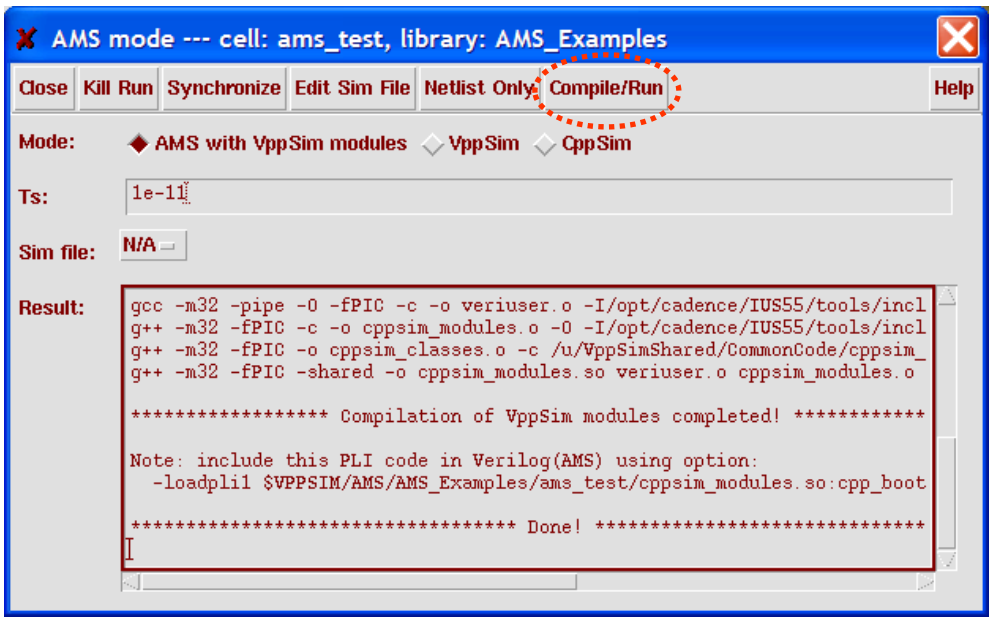


C. Automatic generation of VppSim modules and associated PLI code

- Click on the **AMS with VppSim modules** radio button within the **CppSim/VppSim GUI** window, and then press the **Compile/Run** button.
    - The value of Ts should correspond to the time step associated with #1 within the Verilog portion of the AMS simulation. To explain, VppSim modules execute their code within Verilog always loops that repeat every #1 time step of the Verilog portion of the simulation:

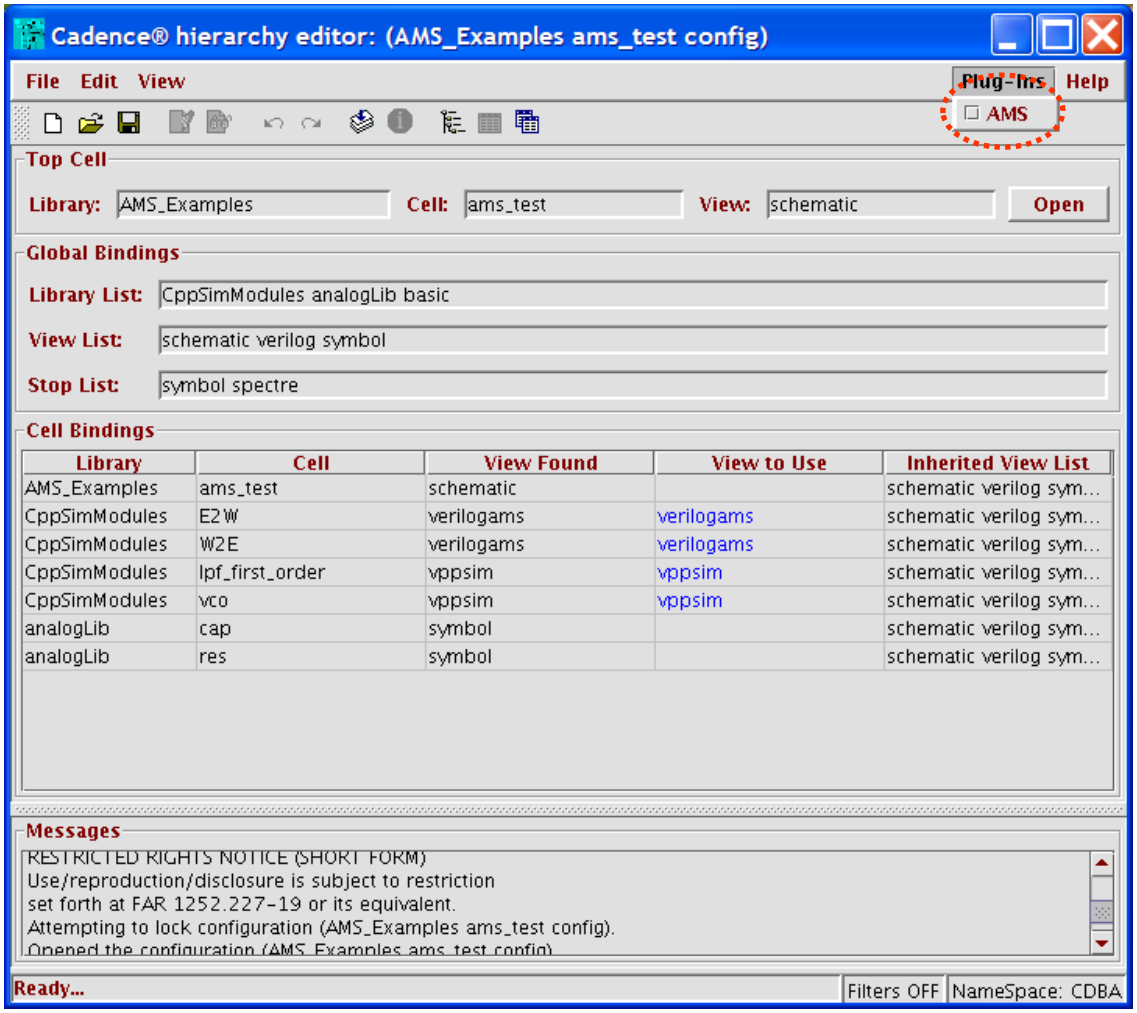
```
always begin
   #1
   $cppsim_module_code(…)
end
```

      For the given example, the default value of 10e-12 is correct (i.e., no need to change the value).
    - Note that the above convention may be changed in the future in order to allow more flexible timing of VppSim modules.
    - At completion of the **Compile/Run** command, **PLI** code corresponding to the CppSim modules within the schematic has been automatically created and compiled. This code will be included in the AMS simulation run, as discussed shortly.

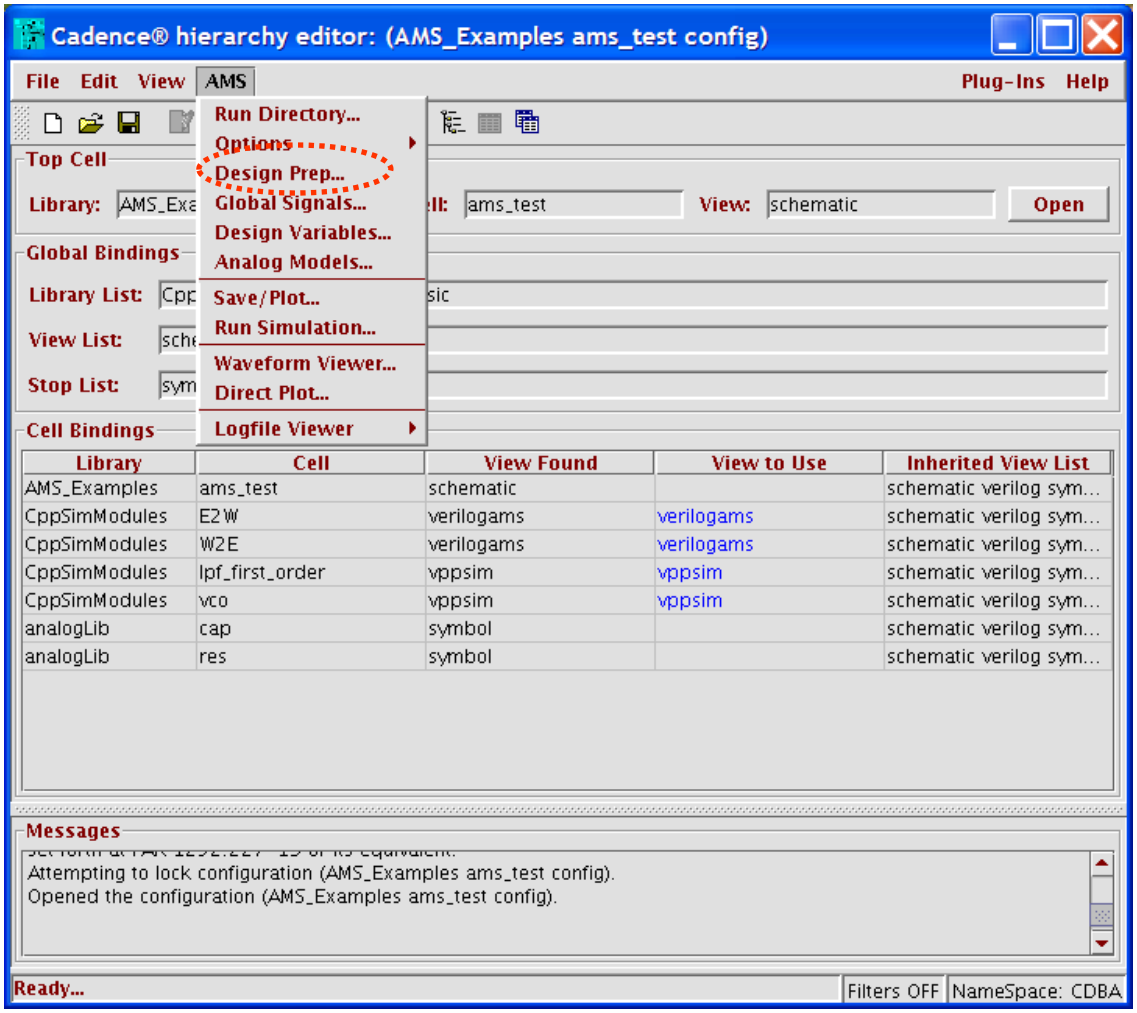


D. Setting up AMS Simulation within the Cadence Hierarchy Editor

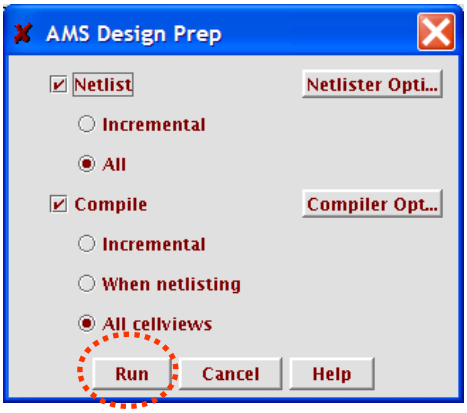
- Click on **Plug-Ins->AMS** to add the **AMS** menu item in the **Cadence hierarchy editor**.



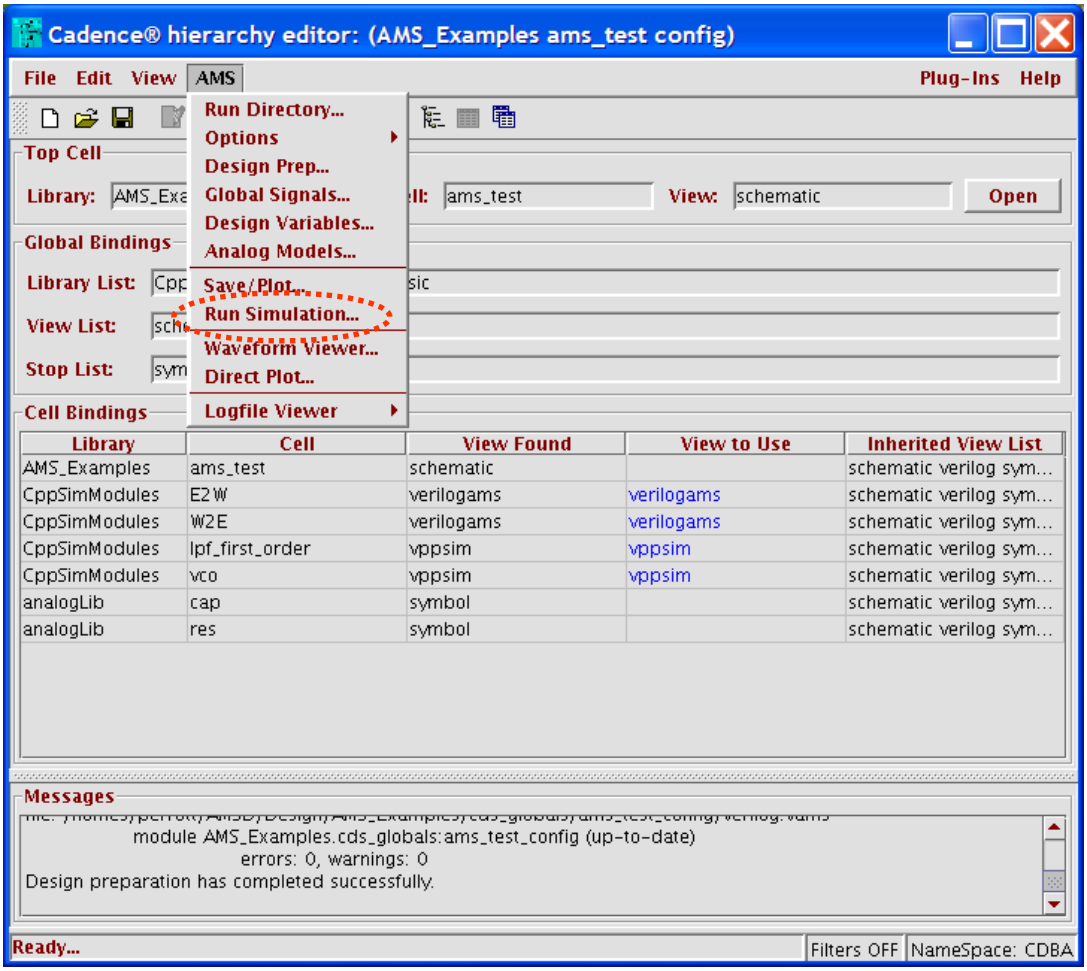
- Click on **AMS->Design Prep…** within the **Cadence hierarchy editor** in order to open the **AMS Design Prep** dialog box.



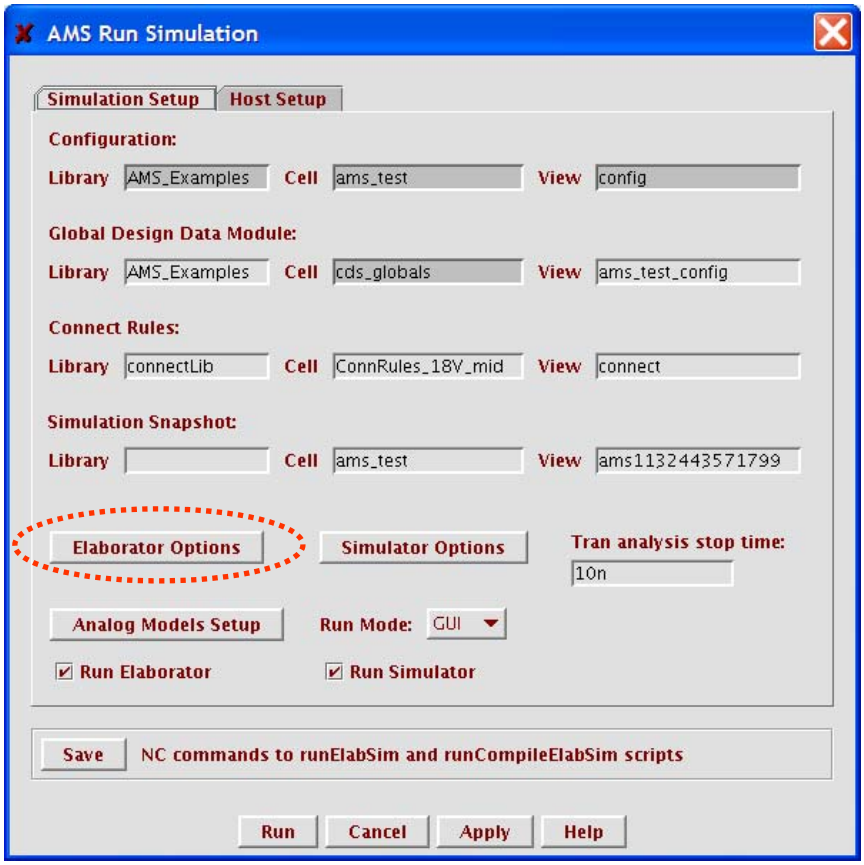
- Click on **Run** in the **AMS Design Prep** dialog box.



- Upon completion of the **Design Prep** step, click on **AMS->Run Simulation…** within the **Cadence hierarchy editor**.



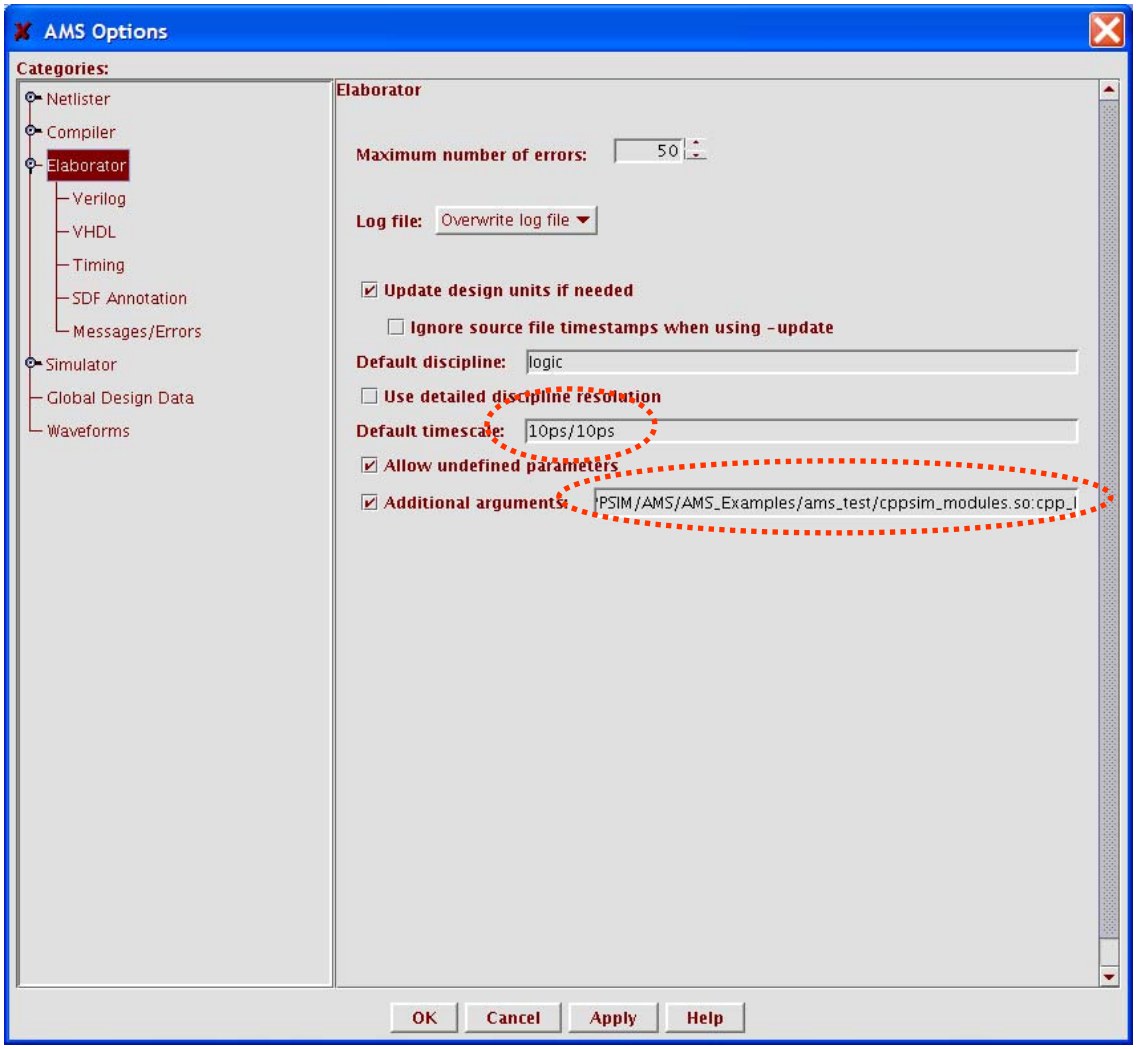
- Click on the **Elaborator Options** button within the **AMS Run Simulation** window that appears.



- Examine the various parameters specified within the AMS Options window that appears. In particular, notice the following items:
    - **Default timescale:** 10ps/10ps
        - This designates that the Verilog portion of the simulation assumes that the #1 time step corresponds to 10e-12. The setting of Ts within the **CppSim/VppSim GUI** window was chosen to match this value.
    - **Additional arguments:**

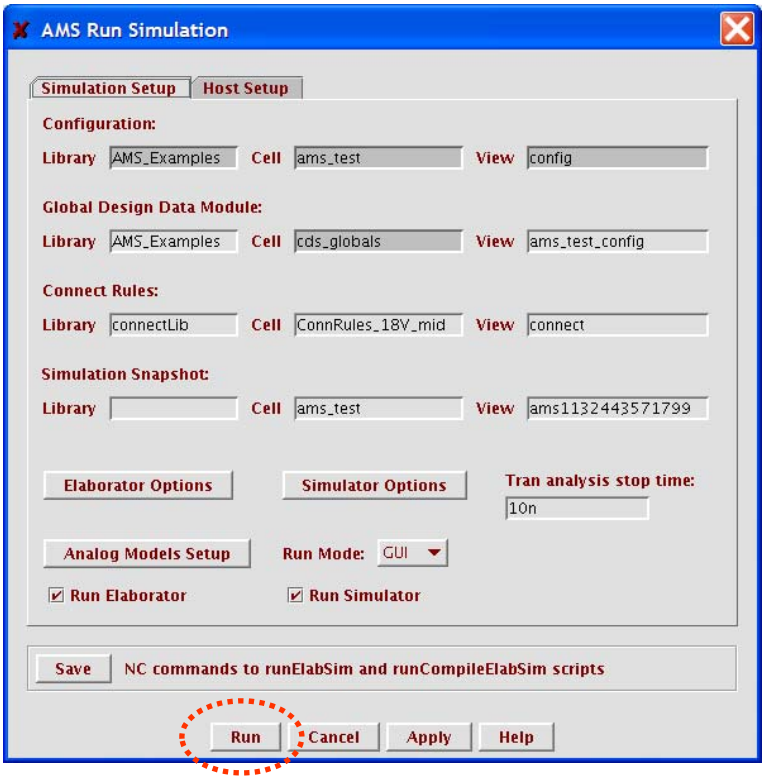
      -loadpli1 $CPPSIMHOME/AMS/AMS_Examples/ams_test/cppsim_modules.so:cpp_bootstrap
        - This includes the PLI code generated from the **CppSim/VppSim GUI** window. Note that the **CppSim/VppSim GUI** window displayed this command so that one could simply paste it into this location if needed.

  Hit **Cancel** in the **AMS Options** window once you have examined the above. All parameters were previously set for you in this case – we simply examined them for informational purposes.



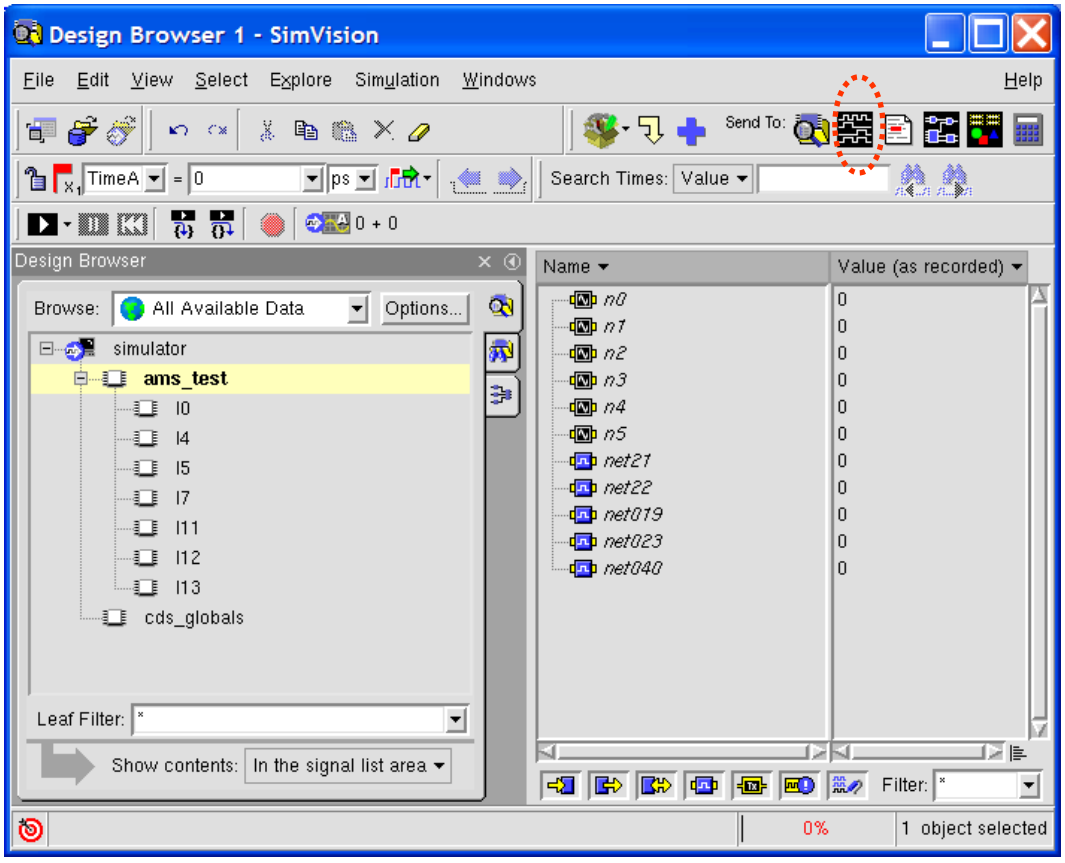
E. Running the AMS Simulation within the Cadence Hierarchy Editor

- Click on the **Run** button in the **AMS Run Simulation** window.

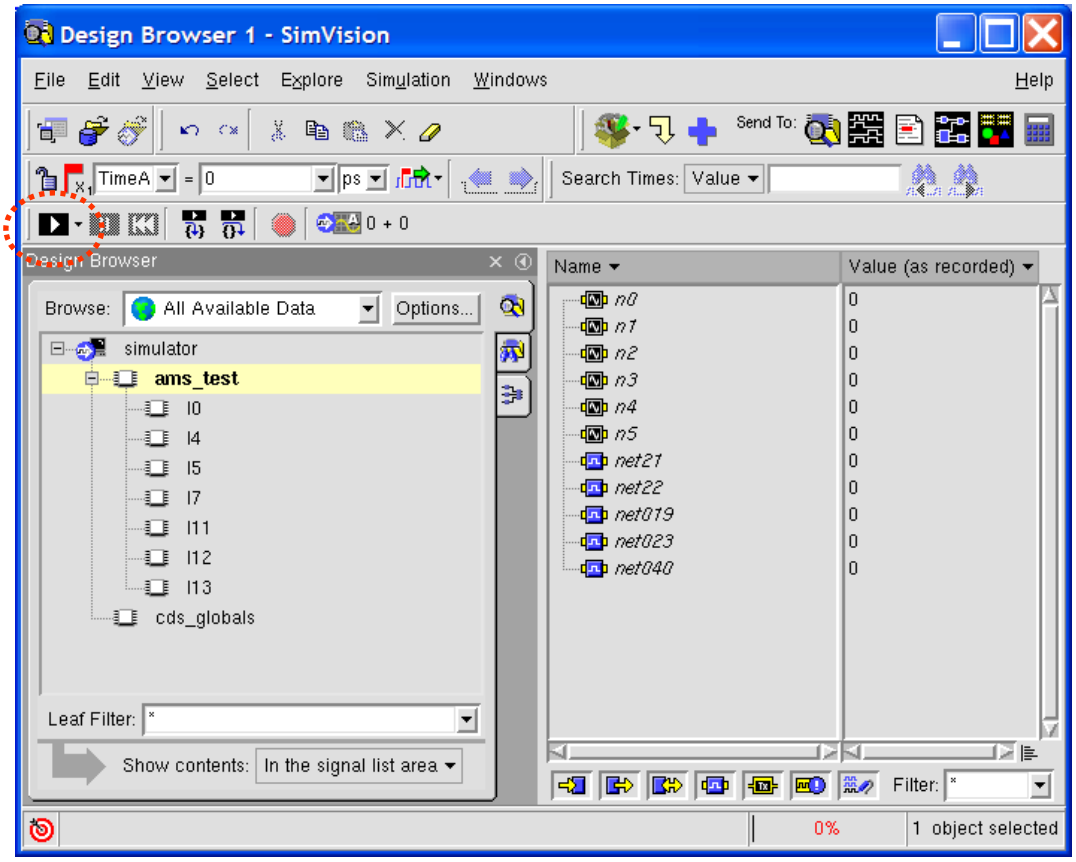


F. Viewing Results within SimVision

- At this point, the **SimVision** windows should appear. Select signals **n0**, **n1**, etc. in the **Design Browser** window by pressing the button highlighted below.



- Click on the button highlighted below to run the simulation.



- After expanding the time axis and fitting the y-axis to the waveforms, the results in the **Waveform** window should appear as shown below.