

Chapter 6

Spectre Analog Simulator

THE SIMULATION described in Chapter 4 is either *behavioral simulation* where the Verilog describes high-level behaviors in a software-like style, or *switch level simulation* where the circuit is expanded all the way to transistors that are modeled as perfect switches using built-in Verilog transistor-switch models. A more accurate simulation would try to model the transistors as analog devices and use as much detail as possible in that analog model to as accurately as possible reflect the real electrical behavior of the transistor network. Analog simulators of this sort can generally trace their background to a simulator called **Spice** that was originally developed at Berkeley in the early 1970's. Through the early 1980's **Spice** was still written in FORTRAN, and you can actually still obtain the **Spice2G6** FORTRAN code from Berkeley if you look around carefully enough. **Spice3** was the first C-language version in 1985. Since escaping from Berkeley there have been many commercial versions of **Spice** and **Spice**-like programs including **Hspice**, **Pspice**, **IS_Spice**, and **Microcap**.

The analog simulator integrated into the **Cadence** framework is called **Spectre**. It is similar to **Spice** in terms of simulating the analog behavior of the transistors, and it even accepts “spice decks” as input in addition to its own **Spectre** format. It operates slightly differently internally and is claimed to be a little faster than **Spice**. From your point of view they are essentially identical simulators though. They take the same input decks and produce the same waveform outputs that show the analog behavior of the circuit network.

An important thing to keep in mind is that while the Verilog simulations used abstract logical 0 and 1 signals for inputs and outputs, analog simulations with **Spectre** will use analog voltages for inputs and outputs. Thus, you now need to be concerned with things like what voltage the power supply is set to, what the voltages should be to be considered a logic 1 or logic

The FORTRAN history of Spice is why even today the input files for analog simulators are called “spice decks.” That term goes back to the punched card decks that were used for input to these FORTRAN programs.

*Note that the simulator described in this chapter is **Spectre** and not **SpectreS** which is an older version with a socket interface. Depending on which version of the NCSU CDK you are using, **SpectreS** may be the default, and you may have to modify the **NCSU_Analog_Parts** library for **Spectre** as described in the appendices.*

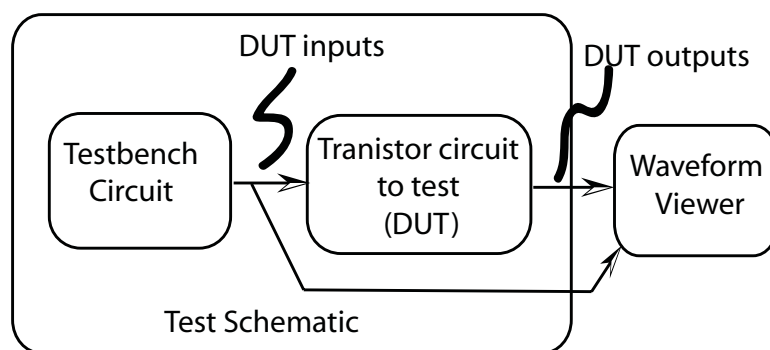


Figure 6.1: The analog simulation environment for a circuit (DUT)

0, what the slope of changing inputs should be, and other analog electronic considerations.

The overall simulation environment for analog simulation of a circuit is very similar to the DUT/testbench model from Chapter 4. The circuit you would like to simulate is the *Device Under Test* or DUT, and you need to construct some sort of testbench circuit around the DUT to provide inputs and sense outputs. The general form of this DUT/testbench approach is shown in Figure 6.1 (compare to Figure 4.1). For analog simulation the inputs need to be described in terms of analog voltages, and the outputs will be returned in terms of analog voltages.

In this chapter we'll see four different ways to use the Spectre simulator as it is integrated with the other Cadence tools. The first three techniques are *transient* simulations of how the circuit behaves over time, and the fourth is a *static* simulation of DC operating points of the circuit. One main difference between these two types of simulation is that in the transient case the inputs and outputs are described as voltages over time, and in the static case the inputs and outputs are defined as static single voltages. In practice these static operating points are swept through a range of voltages to generate curves of, for example, V_{in} versus V_{out} or V_{in} versus I_{out} . The four examples of analog simulation in this chapter are:

1. Transient simulation of a transistor-level schematic described in **Composer**.
2. Transient simulation of a transistor-level schematic where some of the cells have extracted views from the layout and include more information about the physical layout of the transistors and parasitic capacitance from the layout for more accuracy.
3. Transient mixed-mode analog/digital simulation where part of the cir-

cuit is simulated at the switch or behavioral level with a Verilog simulator and part is simulated at the analog level with **Spectre**. This can be very useful for simulating a critical component in the framework of a larger system without simulating the entire system at the analog level.

4. Static DC operating point simulations of individual circuits.

6.1 Simulating a Schematic (Transient Simulation)

To demonstrate the simplest mode of **Spectre** simulation we'll use the NAND gate from Chapter 3, Section 3.3, Figure 3.12. The symbol for this NAND gate was shown in Figure 3.14. To simulate this gate we'll need to open a new schematic and add some analog components so that the simulator will know about the analog environment in which to simulate the NAND. What we're aiming for is to generate analog input waveforms that will stimulate the NAND through all possible input combinations. In Chapter 4 this was done with a testbench Verilog program. For analog simulation we will construct a testbench circuit in **Composer** that includes *voltage sources* to define the power supply and the input signals. Voltage sources are statements in **Spectre** input decks that define voltages in the circuit to be simulated. We will instantiate them as schematic components and they will be extracted into the simulator input deck by the netlister. Commonly used voltage sources in the **NCSU_Analog_Parts** library include:

vdc: A static DC voltage source at a user-defined voltage.

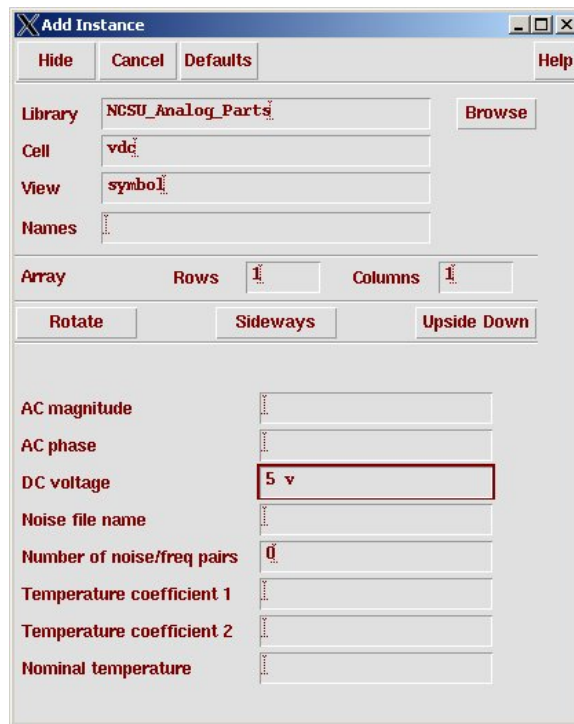
vpulse: A voltage source that generates voltage pulse streams. The user can control aspects of the pulse including delay, pulse width, period, rise time, fall time, and other parameters.

vpwl: A piecewise linear voltage source. The voltage is defined as a set of linear segments that define the output voltage waveform. The syntax is a series of time/voltage pairs that define the vertices of the piecewise linear waveform.

vpwlf: A piecewise linear voltage source that takes its waveform definition (time/voltage pairs) from a file.

vsin: A sine wave voltage generator. The user can set the frequency, amplitude, and other parameters of the output.

vexp: A voltage source that generates a single pulse consisting of an exponentially defined rise and an exponentially defined fall.

Figure 6.2: Component parameters for the **vdc** voltage source

For the NAND example test circuit we'll start with a new schematic, include a copy of the NAND gate (the DUT), and then use **vdc** and **vpulse** voltage sources for the testbench circuit. Open a new schematic, I'll call mine **nand-test**, and add an instance of the NAND gate from your library (or from the **UofU_Digital.v1.1** library if you haven't designed one yet).

*Remember to include an **Asize** frame from **UofU.Sheets** around your schematic!*

*Note that the **vdd** and **gnd** symbols actually connect the nets to the global **vdd!** and **gnd!** labels so that power and ground can be referenced globally in the hierarchy through this one **vdc** connection.*

Now along with this NAND gate you'll need some analog components from the **NCSU_Analog_Parts** library. To let Spectre know about the power supply you need to include a **vdd** and a **gnd** component. These are in the **Supply_Nets** category in the **NCSU_Analog_Parts** library. Go to the **Voltage Sources** category in **NCSU_Analog_Parts** and add a **vdc** DC voltage source. When you instantiate this part, or by using the **q1** hot key later, you should set the DC voltage on this component to be **5 v** as seen in Figure 6.2. The top connection of the **vdc** component should be connected to the **vdd** symbol and the bottom terminal should be connected to the **gnd** symbol. This indicates that for this simulation there should be a five volt power supply (5 v DC between **vdd** and **gnd**).

To generate the input waveforms I'll use two **vpulse** voltage sources. By defining them to have the same pulse width and period, but delaying one of

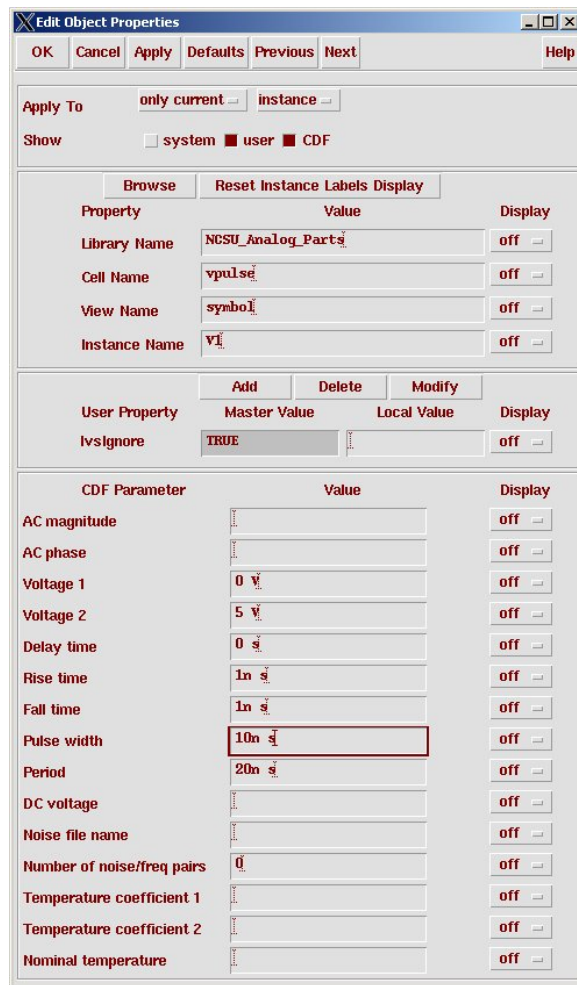


Figure 6.3: Component parameters for the **vpulse** voltage source

the pulses, I can generate all input combinations of high and low voltages to the inputs of the NAND gate. The parameter dialog box for the **vpulse** voltage source is seen in Figure 6.3 where you can see the settings for one of these components. The other **vpulse** component is the same except that it has 5ns of delay before the pulses start.

Finally, I'll connect the NAND gate (DUT) to a capacitor to simulate the effect of driving into a capacitive output load. Another approach to this would be to add other gates to the output of the DUT to simulate driving into specific other gates. I'll add a **cap** component from the **R.L.C** category in the **NCSU_Analog_Parts** library and give it a relatively huge capacitance of 1pf. The final **nand-test** schematic looks like Figure 6.4.

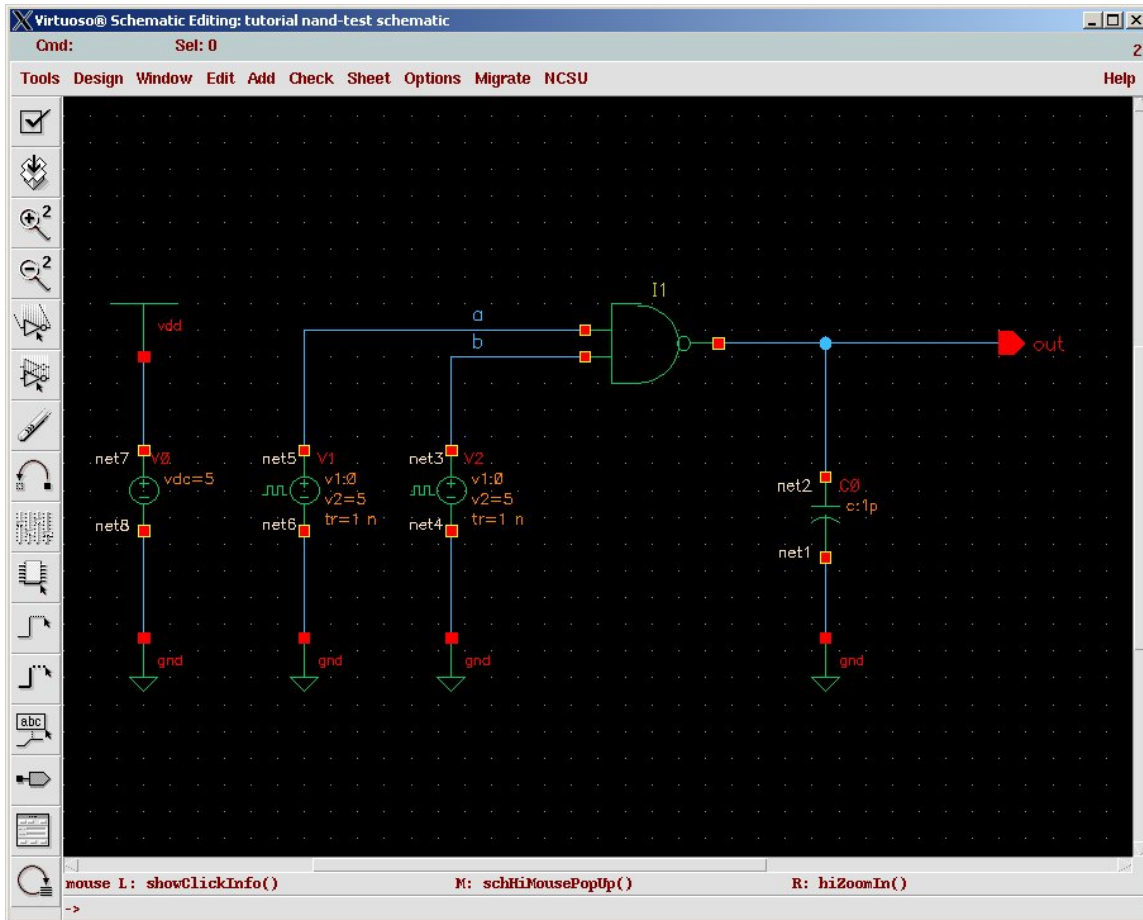


Figure 6.4: Schematic for the nand-test DUT/testbench circuit

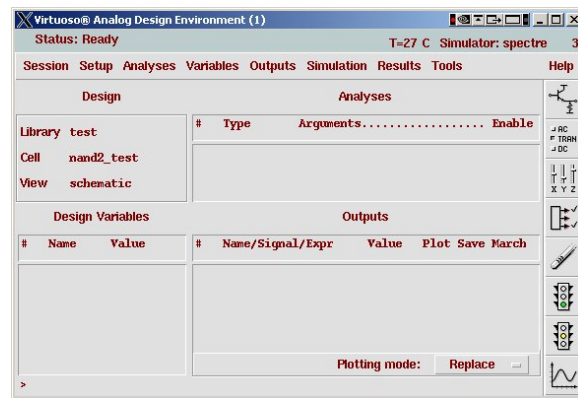


Figure 6.5: Virtuoso Analog Environment control window

6.2 Simulation with the Spectre Analog Environment

Starting with your **nand-test** schematic open in **Composer**, select the **Tools** → **Analog Environment** menu choice. This will bring up a **Virtuoso Analog Design Environment** dialog box as shown in Figure 6.5. You should see the **Library**, **Cell**, and **View** already filled in. If not, or if you'd actually like to simulate a different schematic, you can change this with the **Setup** → **Design** menu in the **Analog Environment**. The other **setup** options should be set for you. We'll be using the **Spectre** simulator, and the **Model Path** should be set to point to the generic nominal transistor models for the class (**ami06.scs** in the class directory). At some point you may wish to point to a different set of models to get models for worst-case (slow) processes corners, or to use models from a specific MOSIS fabrication run. You can set the path to other models that you want to use using the **Setup** → **Model Libraries** menu.

Now select **Analysis** → **Choose...** in the **Analog Environment** or click on the **.** Select **tran** for a transient analysis, and make sure to fill in the **Stop Time** to let the simulator know how long the simulation should run. I'll fill this in to **300n** for 300 nanoseconds of simulation. This dialog box is shown in Figure 6.6.

Now you need to select which circuit nodes you would like to see plotted in the output waveform. The easiest way to do this is to select the nodes that you'd like to have plotted by clicking on them in the schematic. Select the **Outputs** → **To Be Plotted** → **Select on Schematic** menu choice. The **Status** shown in the top of your **Composer** window should now say **Selecting outputs to be plotted...** Select the wires (nodes) that you'd like to see in your output waveform by clicking. I'll select the **a** and **b** inputs to the NAND, and the **out** output nodes. Note that the wires change color in the

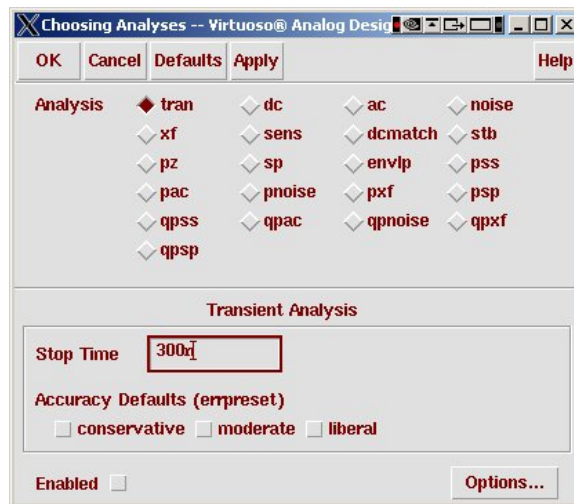


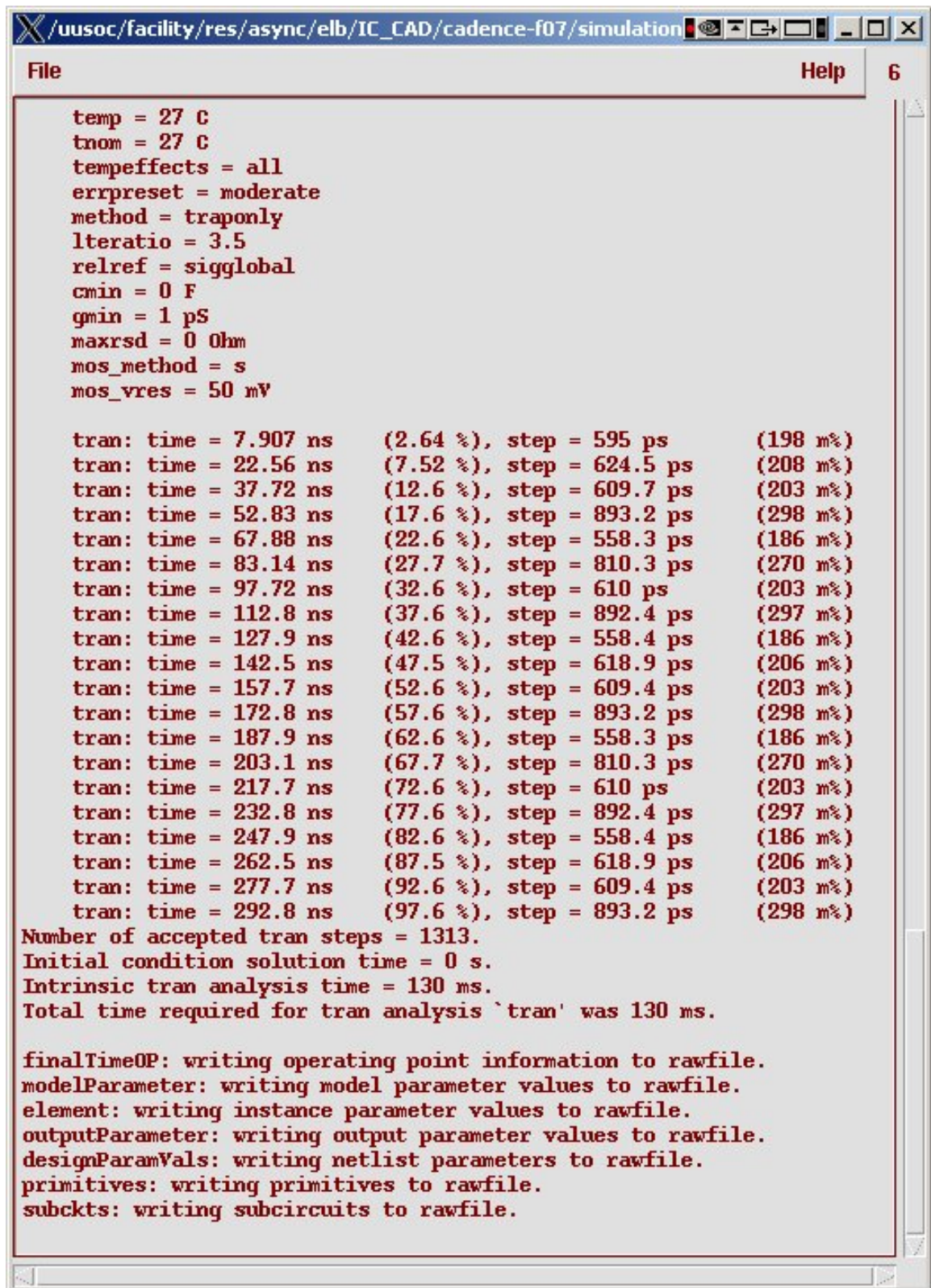
Figure 6.6: Choosing Analysis dialog box

schematic to indicate that they've been selected, and they also appear in the **outputs** pane of the **Analog Design Environment** control window.

*One way to get a runaway simulation is to forget the **n** in the description of the time to simulate. If you're simulating for 300s instead of 300ns the simulation may take a long time!*

Now that you've chosen the design to simulate, the type of simulation (transient, 300ns), and the outputs to be plotted, you can start the simulation by selecting **Simulation** → **Netlist and Run** from the menu, or the **Netlist and Run** widget which looks like a traffic light with a green light. When you run the simulation you'll see the simulation log in the **CIW**. If there are any issues with the simulation the warning and error messages will show up here. If the simulation completes you'll see the elapsed time in the **CIW**, a **Spectre** window that shows the log results of running the simulation, and a waveform window will open up with the results that you specified plotted. The **Spectre** log window looks like that in Figure 6.7.

For our NAND example the waveform window initially looks like that in Figure 6.8. It's a little confusing because all the waveforms are layered on top of each other. This is great if you want to see the details of one waveform in relation to another, but mostly it's just confusing. To fix this you can move to *strip chart mode* by using the **Axis** → **Strips** menu or using the **strip chart mode** widget which looks like four white bars. This will separate the waveforms so that each one is in its own strip. Now the waveform viewer looks like that in Figure 6.9. You can see the inputs in the top two strips (pink and purple in this example). These inputs are generated by the two **vpulse** voltage sources. The output is shown in the bottom strip in yellow. Because of the huge capacitance I put in the **test-nand** schematic the output has a shape characterized by the exponential behavior of an output transistor charging a large capacitance. Using the **Zoom** menu option I can



```
X:\uusoc\facility\res\async\elb\IC_CAD\cadence-f07\simulation
File Help 6

temp = 27 C
tnom = 27 C
tempeffects = all
errpreset = moderate
method = traonly
lteratio = 3.5
relref = sigglobal
cmin = 0 F
gmin = 1 pS
maxrsd = 0 Ohm
mos_method = s
mos_vres = 50 mV

tran: time = 7.907 ns (2.64 %), step = 595 ps (198 m%)
tran: time = 22.56 ns (7.52 %), step = 624.5 ps (208 m%)
tran: time = 37.72 ns (12.6 %), step = 609.7 ps (203 m%)
tran: time = 52.83 ns (17.6 %), step = 893.2 ps (298 m%)
tran: time = 67.88 ns (22.6 %), step = 558.3 ps (186 m%)
tran: time = 83.14 ns (27.7 %), step = 810.3 ps (270 m%)
tran: time = 97.72 ns (32.6 %), step = 610 ps (203 m%)
tran: time = 112.8 ns (37.6 %), step = 892.4 ps (297 m%)
tran: time = 127.9 ns (42.6 %), step = 558.4 ps (186 m%)
tran: time = 142.5 ns (47.5 %), step = 618.9 ps (206 m%)
tran: time = 157.7 ns (52.6 %), step = 609.4 ps (203 m%)
tran: time = 172.8 ns (57.6 %), step = 893.2 ps (298 m%)
tran: time = 187.9 ns (62.6 %), step = 558.3 ps (186 m%)
tran: time = 203.1 ns (67.7 %), step = 810.3 ps (270 m%)
tran: time = 217.7 ns (72.6 %), step = 610 ps (203 m%)
tran: time = 232.8 ns (77.6 %), step = 892.4 ps (297 m%)
tran: time = 247.9 ns (82.6 %), step = 558.4 ps (186 m%)
tran: time = 262.5 ns (87.5 %), step = 618.9 ps (206 m%)
tran: time = 277.7 ns (92.6 %), step = 609.4 ps (203 m%)
tran: time = 292.8 ns (97.6 %), step = 893.2 ps (298 m%)

Number of accepted tran steps = 1313.
Initial condition solution time = 0 s.
Intrinsic tran analysis time = 130 ms.
Total time required for tran analysis `tran' was 130 ms.

finalTimeOP: writing operating point information to rawfile.
modelParameter: writing model parameter values to rawfile.
element: writing instance parameter values to rawfile.
outputParameter: writing output parameter values to rawfile.
designParamVals: writing netlist parameters to rawfile.
primitives: writing primitives to rawfile.
subckts: writing subcircuits to rawfile.
```

DRAFT - Please do not distribute 177
Figure 6.7: Spectre log window for the NAND simulation

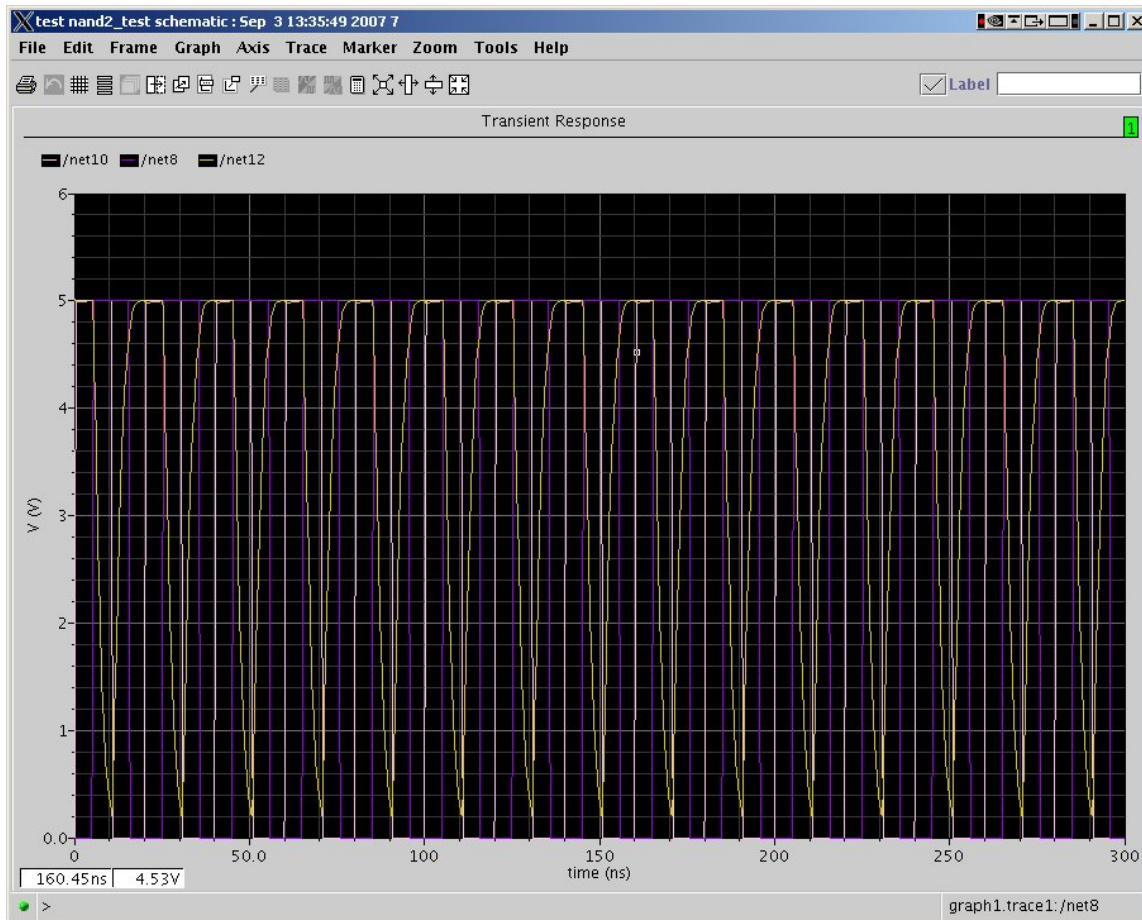


Figure 6.8: Initial Waveform Output Window

zoom in to look at the waveforms more carefully as in Figure 6.10.

You should experiment with the waveform window to see how you can use it to measure and compare waveforms. Some helpful tools are **markers** which are vertical or horizontal cursors that can be moved around in the waveforms and used to measure the curves, and **zoom** to change what part of the waveforms you are looking at.

You can place markers in the schematic using the **Marker** → **Place** → **Vert Marker** or **Horiz Marker** menus. Markers are vertical or horizontal lines that you can place in the schematic which will measure points on the waveforms as you move them around. As an example I'll place two vertical markers in the waveform. These markers will be labeled **M0** and **M1**. You can move them around with the left mouse button (the cursor changes to <> when you hover near them). Make sure that when you select a location

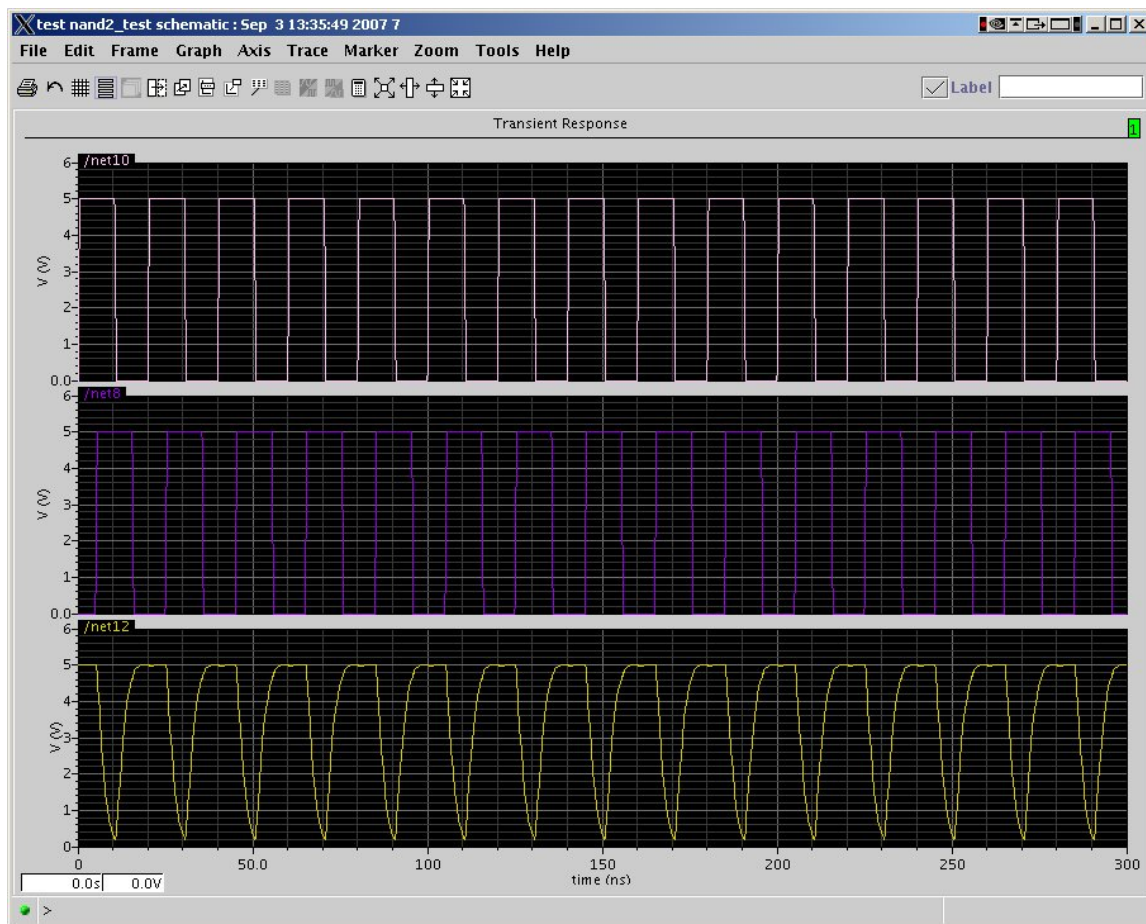


Figure 6.9: Waveform Output Window in Strip Mode

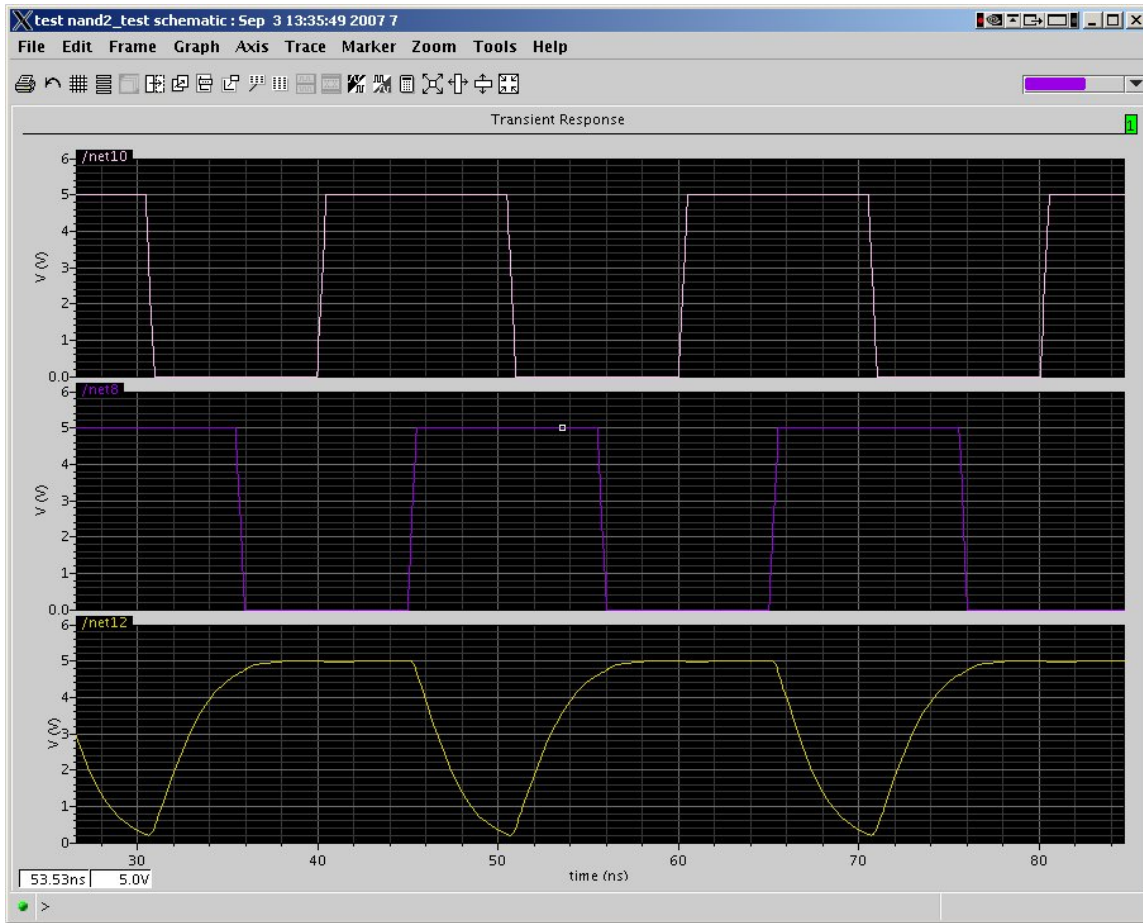


Figure 6.10: Waveform Output Window: Zoomed View

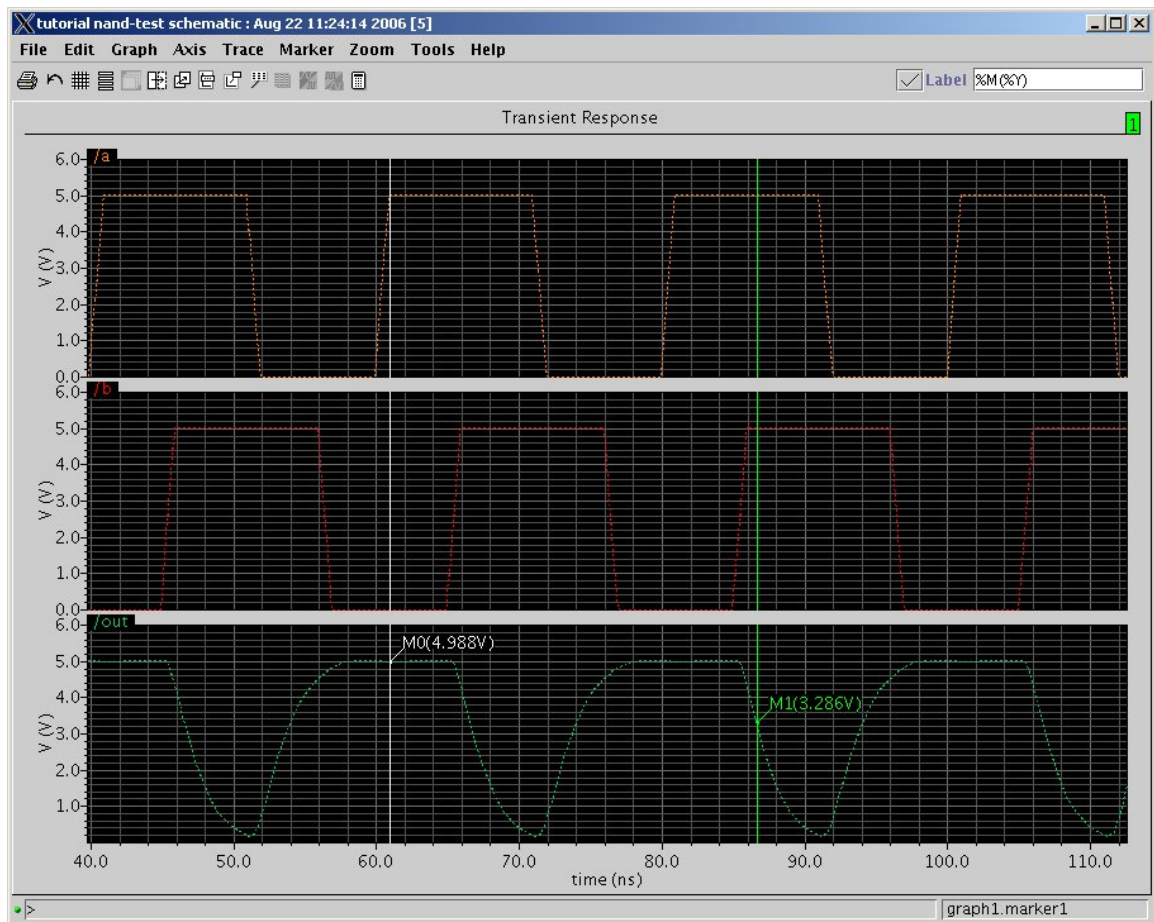


Figure 6.11: Waveform Output with Markers

for the marker that you select inside the trace that you want to see. This will attach the vertical voltage marker on that trace. If you want to change the trace that a marker is associated with you can pick up the trace, move it off the right side of the screen, and then move back into the trace you want to see. Figure 6.11 shows the trace with two output markers inserted. You can use both horizontal and vertical markers to measure the waveforms.

6.3 Simulating with a Config View

In the previous section, the NAND gate was simulated by making a separate schematic that included the NAND (the DUT) and a testbench circuit around that NAND instance, then simulating that schematic with Spectre. This works very well if what you want to simulate is the **schematic** (or

*Recall that the **analog.extracted** view is generated only after LVS succeeds! See Chapter 5, Section 5.6.1.*

*The **analog_extracted** view is essentially the same as the **extracted** view, but with additional information about power supply connections for simulation.*

cmos_sch) view of the NAND. That is, the schematic that contains the transistor level netlist. But, as you've seen in Chapter 5, you can also have a **layout** view, and from the layout view you can generate an **extracted** and an **analog_extracted** view of the same cell. The **analog_extracted** view has additional information about the circuit that is useful for simulation such as parasitic capacitance of the wires, exact layout dimensions of the transistors, etc. In order to simulate the **analog_extracted** view you need a way to tell Spectre (through the **Analog Environment**) which view you really want to simulate.

Imagine that you have a **nand-test** schematic like that shown in Figure 6.4. If you make that schematic into yet another view, called a **config** view, then you can use that **config** view to select which underlying view of the NAND you'd like to simulate: the **cmos_sch** or the **analog_extracted**.

One way to do this is by making yet another view called a **config** view of the testbench cell. To make a **config** view you need a **schematic** view to start with. I'll use the existing **nand-test** schematic. Now, in the **Library Manager** select **File** → **New** → **Cell View**. Fill in the **Cell Name** as **nand-test** and the **View Name** as **config**. This should automatically select **Hierarchy-Editor** as the **Tool**. See Figure 6.12. In the **New Configuration** window that pops up select **Use Template**. In the **Use Template** dialog select **Spectre** as the template name and click **OK**. Back in the **New Configuration** dialog you will see that the **Global Bindings** and other fields have been filled in. Make sure that the **Library** and **Cell** are filled in with your schematic name, and change the **View** from **myView** to **schematic**. Figure 6.13 shows this dialog after this has been done. Click **OK** to create the **config** view.

What you see for the **config** view is the cell described in the **Hierarchy Editor**. The initial window is seen in Figure 6.14. You can see that the **nand-test** cell is defined by five components from the **NCSU_Analog_Parts** library (**cap**, **nmos**, **pmos**, **vdc**, and **vpulse**) that have **SpectreS** views), a **nand2** component from my **UofU_Example** library (with a **cmos_sch** view), and the top-level **nand-test** schematic (a **schematic** view). This is just a list of all the cells in the hierarchy. A better view is to change the view to the **tree** view using **View** → **tree**. The **Hierarchy Editor** window with the **tree** view is shown in Figure 6.15.

In the **tree** view you can see each of the components at the top level, and by expanding the **nand2** component you can see the transistors that are in that component's level of the hierarchy. Now select the **nand2** component. Right click on **nand2** and in the pop-up menu select **Set Instance View** → **analog_extracted**. In the **Hierarchy Editor** window you now see that the **View to Use** field of the **nand2** component now says **analog_extracted** in blue. Update the **config** view with **View** → **Update**, **File** → **Save** and you

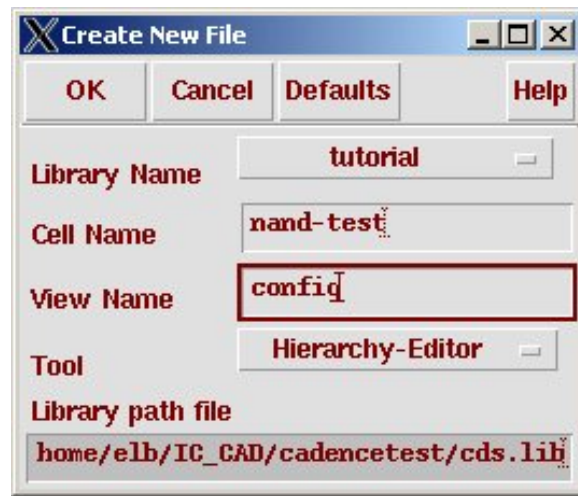


Figure 6.12: **Create New File** dialog for the **config** view

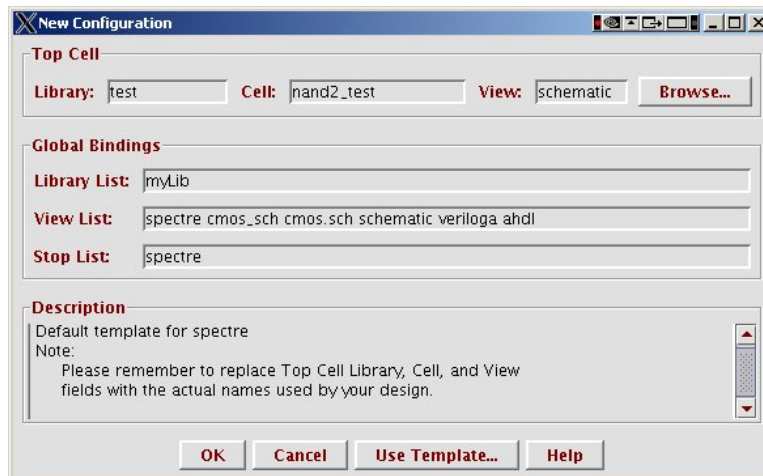


Figure 6.13: **New Configuration** dialog box

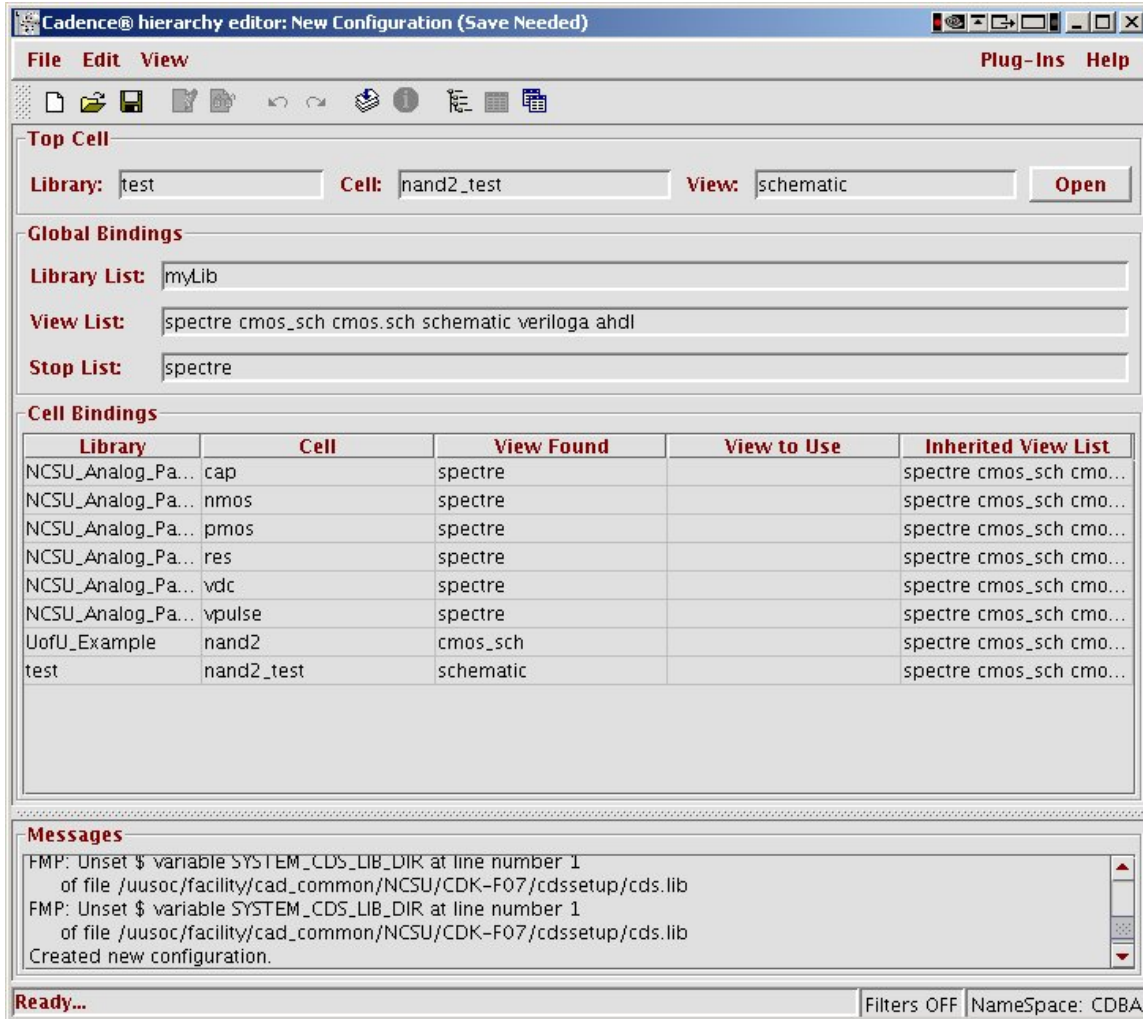


Figure 6.14: Hierarchy Editor view for **nand-test** (table view)

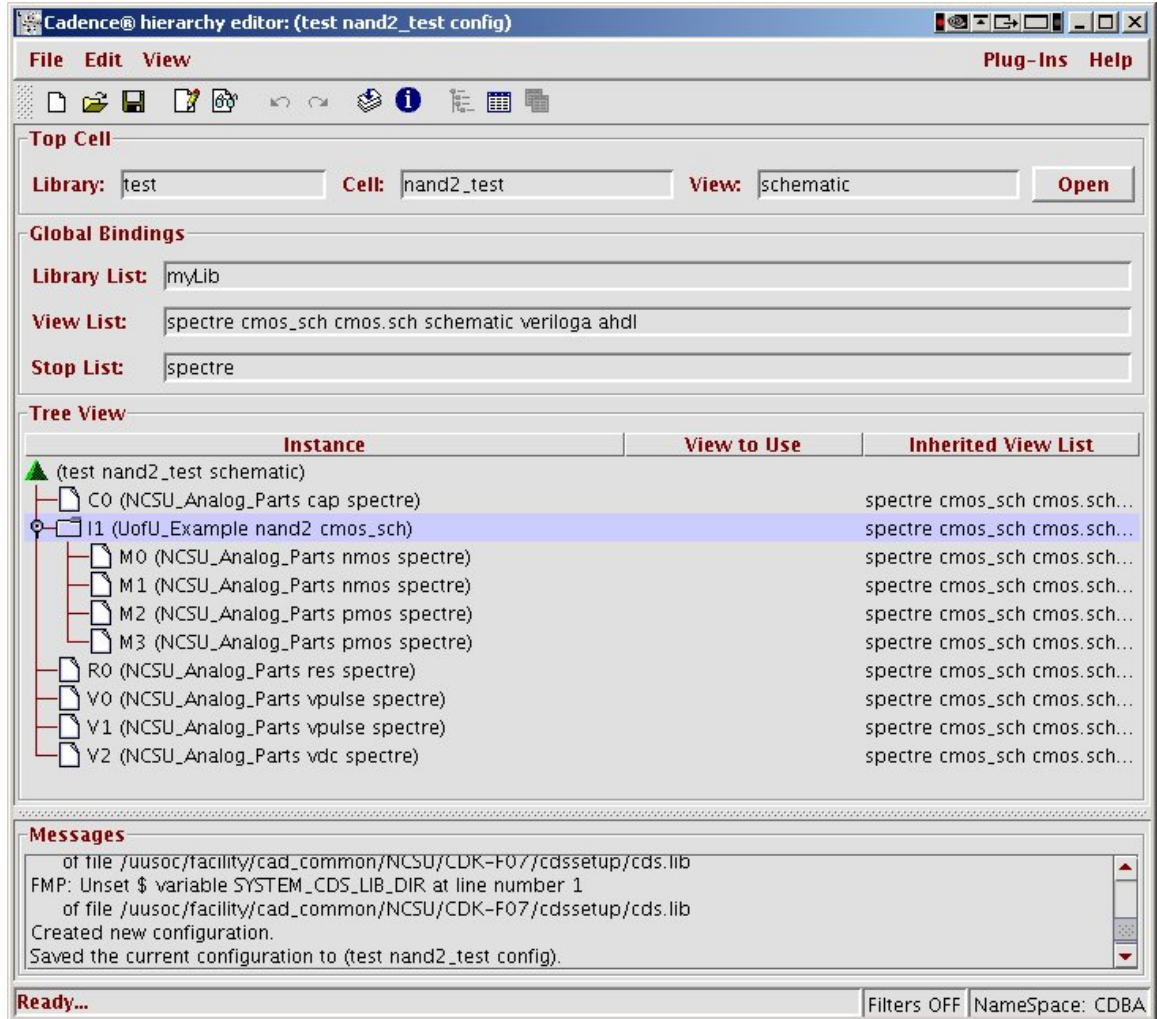


Figure 6.15: Hierarchy Editor view for **nand-test** (tree view)

have saved a new view that will use the **analog_extracted** view instead of the **cmos_sch** view when you use Spectre.

After closing the **Hierarchy editor** I now have two views of my **nand-test** cell: **schematic** and **config**. If I double click on the **config** view to open it, I get a choice of whether to open the **schematic** along with it. If I do this, and select **Tools** → **Analog Environment** as I did in the previous section, you can see that the **Analog Design Environment** dialog opens up just like in the last section (Figure 6.5), but this time the **View** is set to be **config**. If you follow the same steps for simulation you will get the same waveform output, but this time it will have been the **analog_extracted** view that was simulated because that's the view you specified in the **config** view through the **Hierarchy editor**. You can use this technique in a large schematic to select any combination of (simulatable) views that you want for each of the components. For this example it makes almost no difference in the simulation results, but for a larger circuit the difference can be significant because of the additional information in the **analog_extracted** view that is not in the **cmos_sch** view.

6.4 Mixed Analog/Digital Simulation

The most detailed and accurate simulation in this flow is analog simulation using Spectre. This very detailed simulation of every transistor in your design gives you timing results that are within a few percent of the fabricated chip. Of course, it's also very slow, especially for large chips. It's also difficult to use the **vpulse** and **vpwl** components to generate complex data streams for digital circuits. However, if you really want good timing information about your chip, there's no substitute for analog simulation of the whole chip!

Even more detailed simulation is possible that includes 3-d transistor and interconnect models, field-solvers for understanding the effects of changing signals, localized heating, and many other effects, but they're beyond the scope of this flow.

Luckily, there is a compromise between full analog simulation and purely functional simulation. Cadence is designed to do mixed mode simulation where part of your design is simulated using Verilog-XL or NC_Verilog and part is simulated using Spectre. You can use this capability for a variety of simulation tasks.

1. You can simulate circuits that are actually mixed mode circuits. That is, systems that have both analog and digital components like a successive approximation ADC.
2. You can simulate large digital systems by simulating most of the circuit using a Verilog simulator, but specify that certain critical parts are simulated using the analog simulator for more accuracy.

3. You can simulate the entire system with the analog simulator, but have a small set of digital components in your testbench file so that you can write the testbench in Verilog instead of using **vpulse** and **vpwl** components.

Cadence does this sort of simulation through the **config** view that was described in the previous section. It also automatically installs *interface elements* between the digital and analog portions of the design (they're called **a2d** and **d2a**), and automatically interfaces the two simulators.

To set up for mixed mode simulation you need a top-level schematic that includes your DUT, and also components that drive the input and deliver the outputs. In the top-level schematic for the pure **Spectre** case the inputs were driven by analog voltage sources. Because we want to drive the inputs from Verilog in this case, the inputs should be driven by digital sources. Because the mixed-mode simulator wants to include interface elements between the digital and analog parts of the circuit, I'm going to include two inverters in a row driving the inputs, and two inverters in a row for the output signals. This will let me make one of these inverters digital (so that it can be driven from Verilog) and the second inverter can be analog (so that it will provide an analog signal to the NAND gate DUT). If there are inputs to your DUT that you want to be driven from analog voltage sources you can also include those in this test schematic. Because part of the simulation is analog, you also need to include the **vdc** for the power supply connection in any case. The **mixed-test** schematic is shown in Figure 6.16.

Create a config view of your testbench schematic as if you were doing an analog simulation. When you get to the **New Configuration** dialog box, use a template to set things up. To set up for mixed mode simulation use the **SpectreVerilog** template (remember to change the view to schematic). The main difference from this **config** view and the **config** view used for pure **Spectre** is that the **View List** and **Stop List** include some Verilog views because part of the circuit will be simulated as Verilog.

Now you have a config view that describes each instance in the schematic and what view to use to simulate it. The trick to mixed mode simulation is to specify a view for some cells that results in analog simulation, and a view for other cells that results in digital (Verilog) simulation for those cells. Note that you're making this choice for a tree of cells. If you make a choice for a cell, all cells under that cell inherit that choice unless you descend into the hierarchy and override that decision. From the initial **Hierarchy Editor** view (Figure 6.17) you can see that every component has the **behavioral** view selected initially so the entire simulation will be digital Verilog simulation (this is the **tree view** of the **config** view).

Before we change the partitioning of the circuit to make some of the

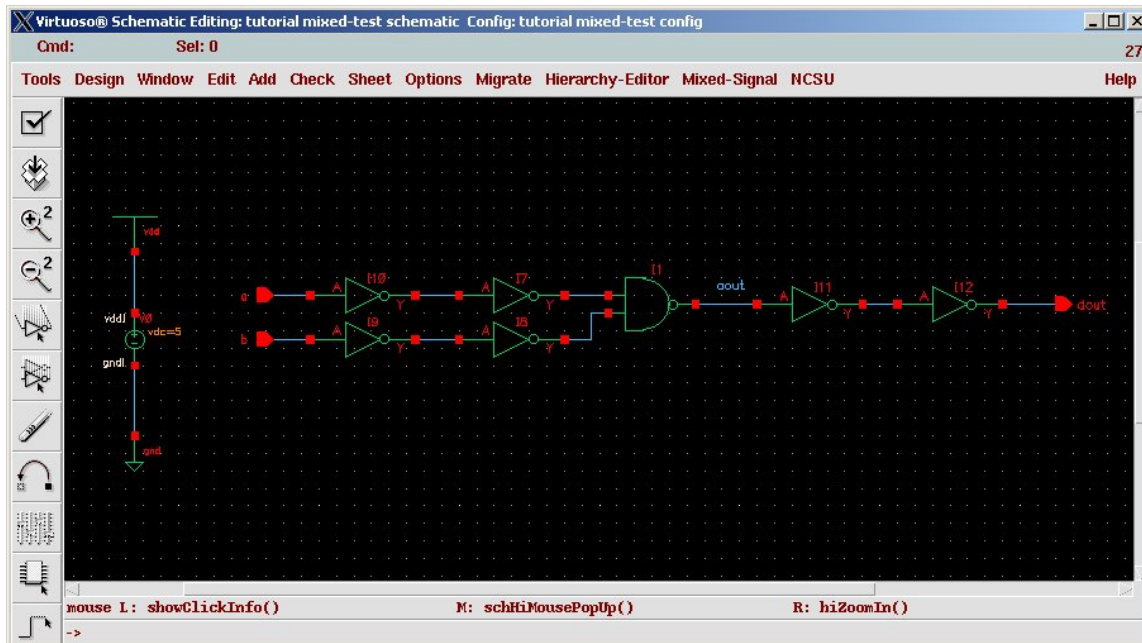


Figure 6.16: Test schematic for the mixed-mode NAND (DUT) simulation

components simulated by Spectre we have to make sure that the mixed-mode simulator can find the interface elements that it wants to put between the digital and analog portions of the circuit. If you have just created the **config** view you'll need to close the view, and re-open the same **config** view from the library manager. This time if you select **yes** for both the **config** and **schematic** views you'll get both windows at the same time. Switch the **Hierarchy Editor** to **tree** view, and in the schematic window select **Tools** → **Mixed Signal Ops** to get a few new menu choices in the Composer window.

In the Composer schematic window select the **Mixed-Signal** → **Interface Elements** → **Default Options** menu choice to get the dialog box. In this box, update the **Default IE Library Name** to be **NCSU_Analog_Parts** (see Figure 6.18). Now you can select **Mixed-Signal** → **Interface Elements** → **Library** to change the default characteristics of the interface elements. For example, the **output d2a** devices have rising and falling slopes, and high and low voltages defined so that they know how to take digital signals and generate analog versions. For the **input** devices you can set at what analog voltage the **a2d** thinks the value is a logic 0 or a logic 1, and the max amount of time an **a2d** can remain between a logic 1 and logic 0 before it reports an X to the digital simulator. The defaults will probably work fine, but this is where you change things if you want to. Figures 6.19

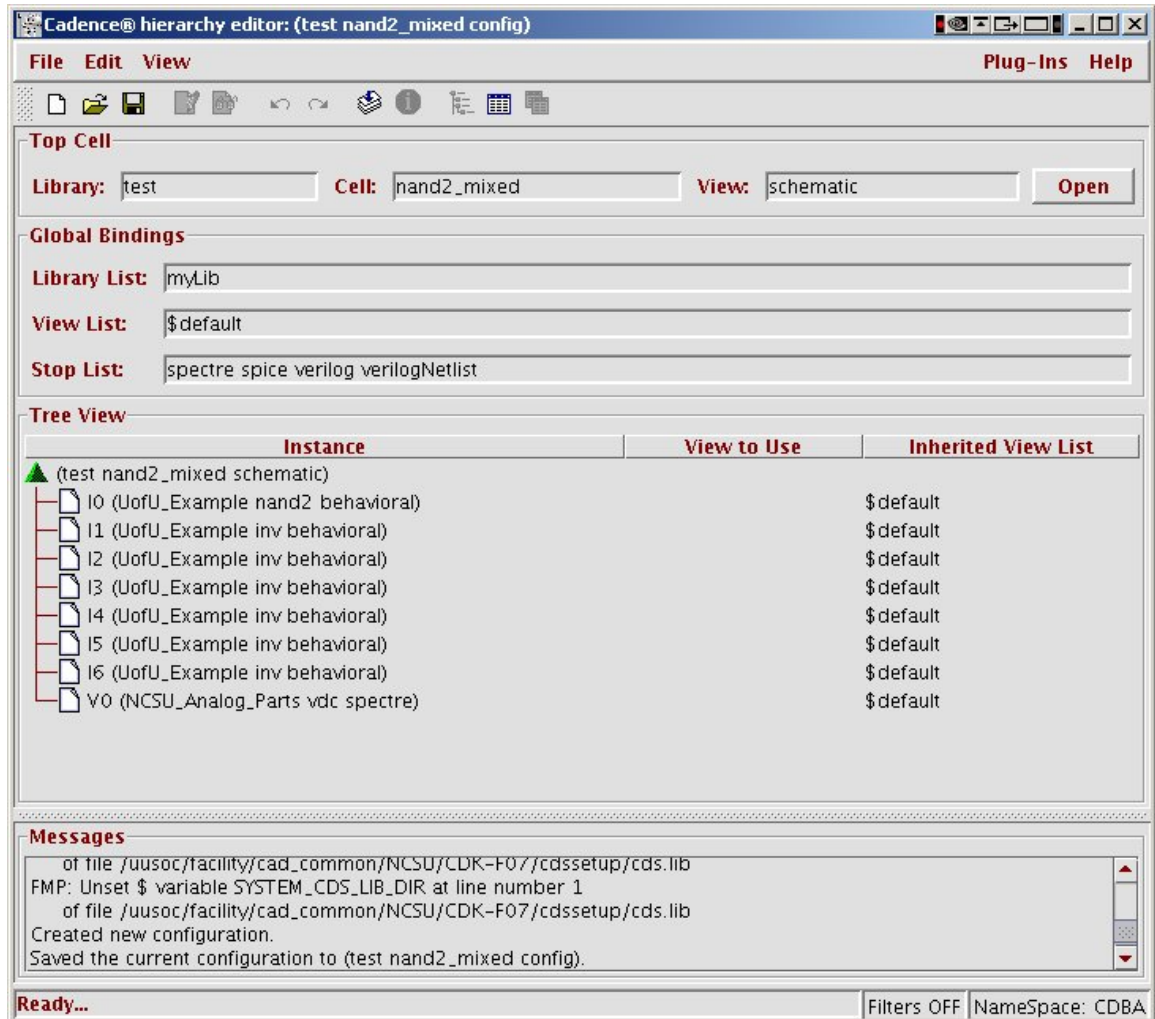


Figure 6.17: Mixed mode config view for mixed-nand

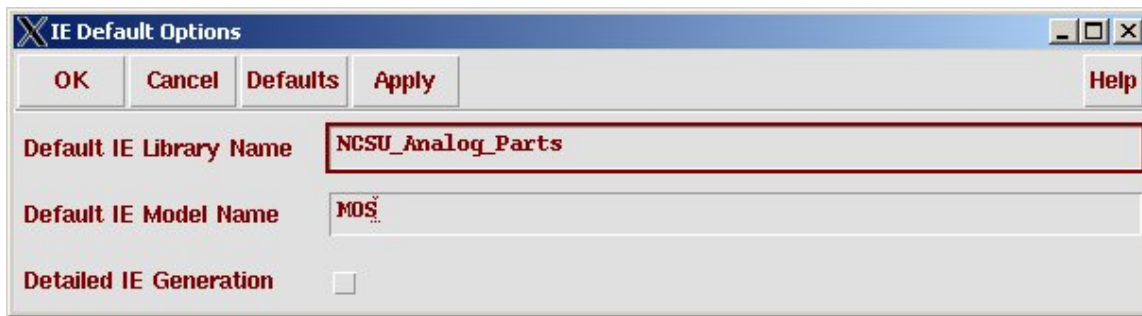


Figure 6.18: Interface Library Dialog Box

and Figure 6.20 show the default values.

Once you have things set up, you can change the partitioning by changing how the **config** view looks at each circuit. In this case I'll change the inverters closest to the DUT (the NAND gate) to use analog simulation, and choose analog simulation for DUT itself machine. I'll be able to apply inputs to the test circuit using Verilog to drive the **a** and **b** signals. I can see the analog output on **aout** and the digital output as the output of the second output inverter (which is simulated with Verilog) **dout**.

*Note that if you select the inverters in the **config** view they are also highlighted in the **schematic** view.*

In my schematic, inverters I1, I4 and I5 are the ones closest to the DUT (NAND). I'll change their **View to Use** to **cmos_sch** (using a right-mouse and **Set Instance View**) so that the netlister will expand them, and change their inherited view list to be **spectre** so that the netlister will stop only when it finds that view (which is an analog view according to the Analog Stop List). The **View to Use** is updated by right-clicking the component and using the menu. The **Inherited View List** is updated by clicking on the field and then typing **spectre** by hand.

I'll leave the other inverters as is (using the **behavioral** view) so that they'll be simulated by the Verilog simulator. For the **nand2** DUT block, I'll change it to a **analog-extracted** view, and **spectre** for the **Inherited View List** so that it's also simulated with **spectre**, but with the extracted parasitics. Note that if the DUT had more structure, that is it was composed of a hierarchy of **schematic** and **cmos_sch** views I would need **schematic**, **cmos_sch** and **spectre** in the **Inherited View List** so that the netlister will continue to expand the schematic views until it finally gets to a **spectre** stopping view. The config now look like Figure 6.21. Update the view using **View** → **Update**, and save the **config** view before moving on.

You can now go back to the schematic view and click on **mixed-Signal** → **Display Partition** → **All Active** to see the partitioning that you've spec-

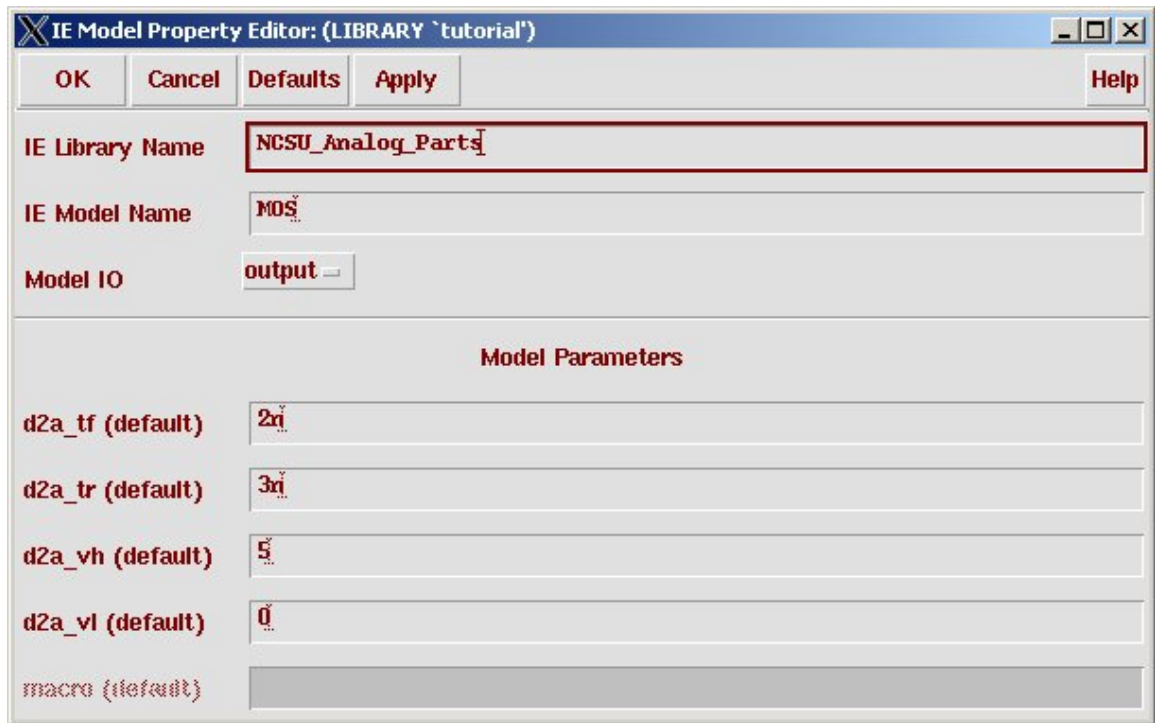
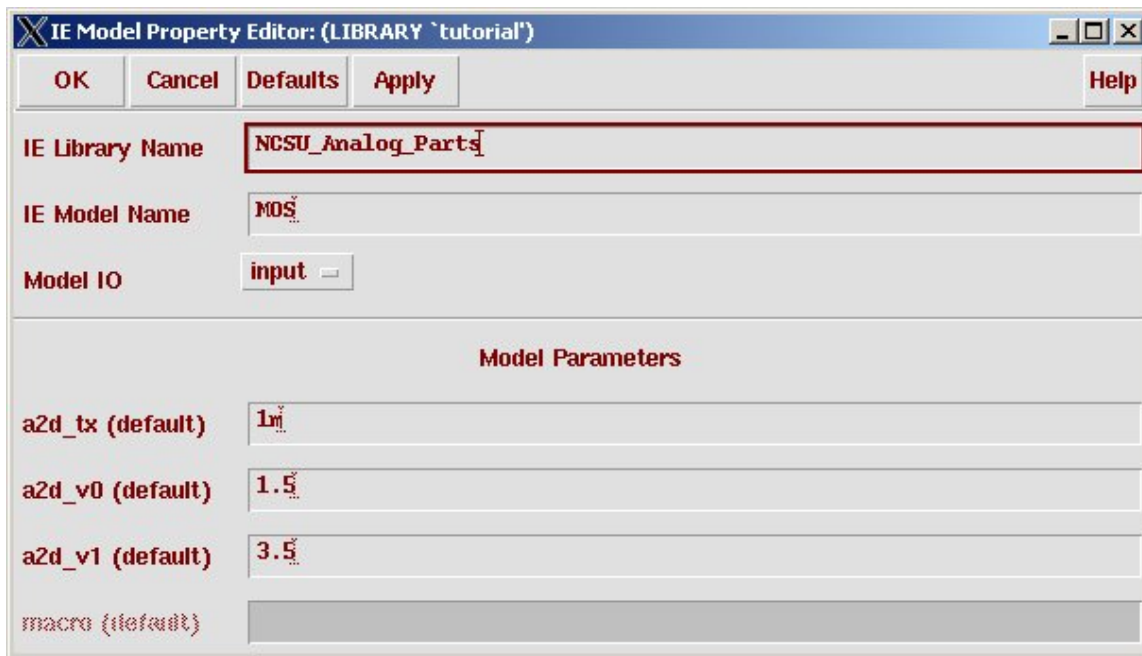


Figure 6.19: **d2a** interface element parameters

Figure 6.20: **a2d** interface element parameters

ified. As you can see in the schematic (Figure 6.22), the components highlighted in orange will be simulated with Verilog and the components highlighted in red will be simulated with Spectre. The mixed-signal support process has added **d2a** and **a2d** components between the simulation domains for you (shown in blue in the schematic view).

Now in the Composer schematic you can select **Tools** → **Analog Environment** to start the mixed mode simulation. This will look just like the dialog box in Figure 6.5 but will have **config** as the **View** to simulate. Before you start the rest of the analog simulation process, you need to change which simulator will be used. Select **Setup** → **Simulator/Directory/Host** and change the **Simulator** to **spectreVerilog** so that the analog environment knows that you're going to use both analog and Verilog simulators. If you're using different transistor models than the class defaults make sure you update the **model path** with the **Setup** → **Model Libraries** menu. Also, you may want to check in the **Setup** → **Environment** menu, in the **Verilog Netlist Option** sub-form, make sure you're using **Verimix** as the **Generate Test Fixture Template** type and that **Drop Port Range**, **Netlist SwitchRC** and **Preserve Buses** are all selected. These should be selected by default but it's worth a minute to check.

Now you need to edit your digital testbench code (as you did in Chap-

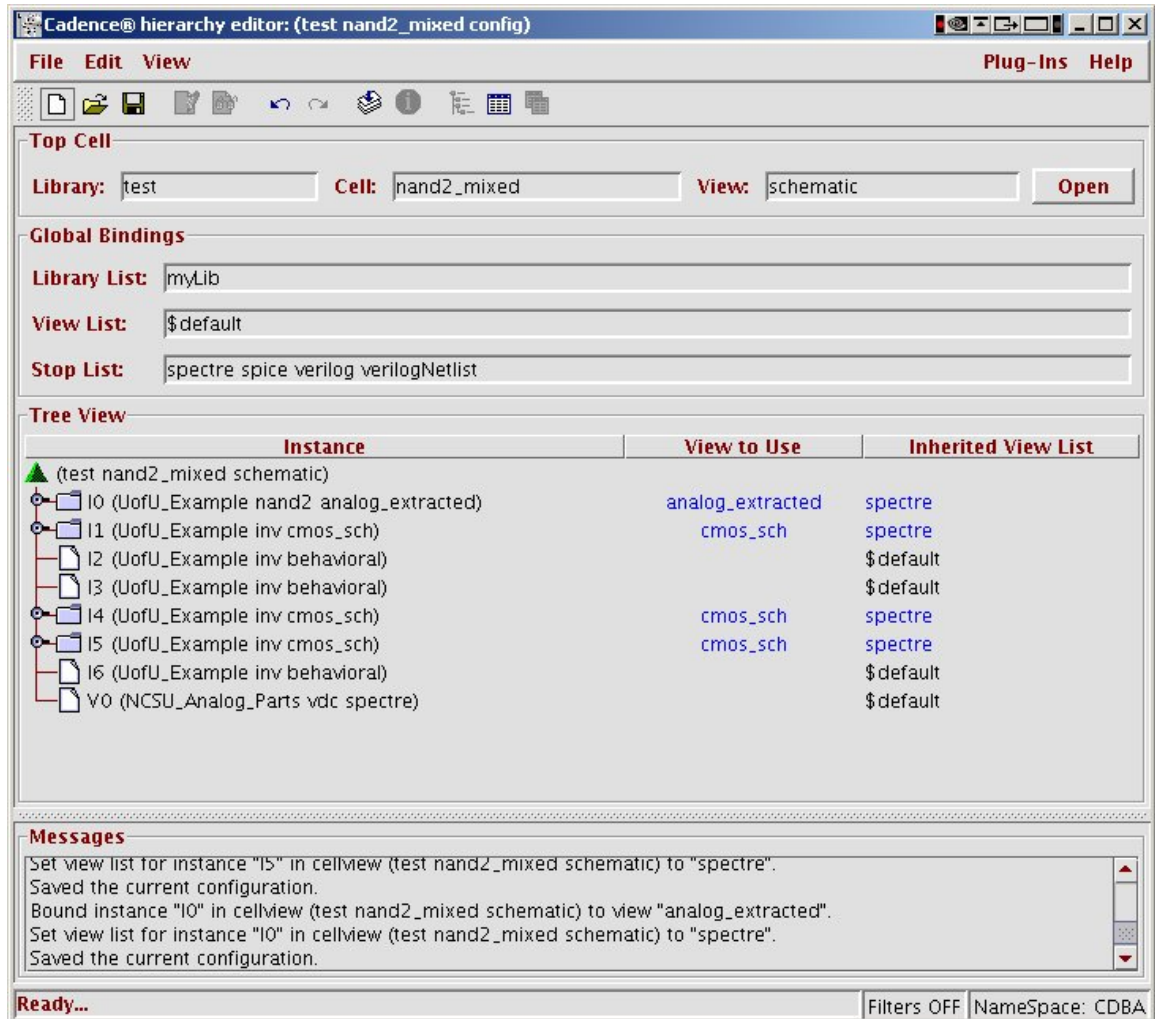


Figure 6.21: Mixed-mode **config** view with analog/Verilog partitioning

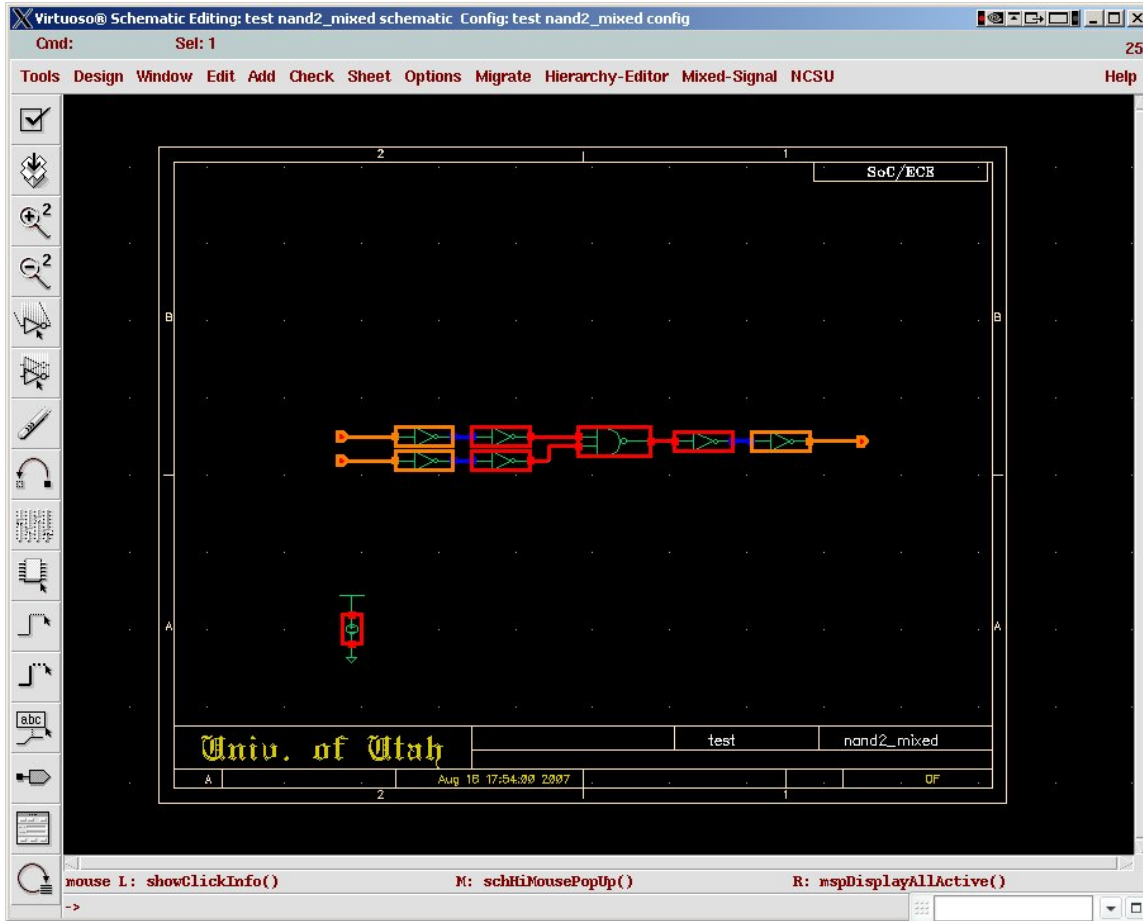


Figure 6.22: The **mixed-test** schematic showing analog/digital partitioning

```

// Vermix stimulus file.
// Default verimix stimulus.

initial
begin

    a = 1'b0;
    b = 1'b0;

#10 $display("ab = %b%b, out = %b", a, b, dout);
if (dout != 1) $display("Error - that's wrong!");

a=1;
#10 $display("ab = %b%b, out = %b", a, b, dout);
if (dout != 1) $display("Error - that's wrong!");

b=1;
#10 $display("ab = %b%b, out = %b", a, b, dout);
if (dout != 0) $display("Error - that's wrong!");

a=0;
#10 $display("ab = %b%b, out = %b", a, b, dout);
if (dout != 1) $display("Error - that's wrong!");

end

```

Figure 6.23: The digital testbench for the mixed-nand simulation

ter 4). Select **Setup** → **Stimului** → **Digital** to make your Verilog Testbench. My example testbench is shown in Figure 6.23. It looks just like a pure Verilog testbench, but the “guts” of the simulation of the DUT (the NAND gate) will have been done in the Spectre analog simulator.

This testbench includes Verilog delays that in total last for 40ns so the transient analysis in Spectre had better go for at least that long. I’ll choose a transient analysis of 100ns for this example just to be safe. I’ll choose analog signals to be plotted by selecting them on the schematic just like in the previous cases, but for fun I’ll also select the digital inputs and outputs. Then I can start the simulation. After switching the display to **Strip Chart Mode** you can see in Figure 6.24 that the digital waveforms are output at the top, and the analog waveforms are plotted in the bottom three strips. By clicking and dragging the digital waveforms over the top of the analog waveforms I can also generate a waveform like that in Figure 6.25 where the digital waveforms are superimposed on the analog waveforms. In this Figure you can see that the analog inputs to the DUT are delayed because they have to go through two inverters, and the analog output is delayed through a pair of inverters before it becomes the digital output.

The next question you might have is: Where did the **\$display** outputs

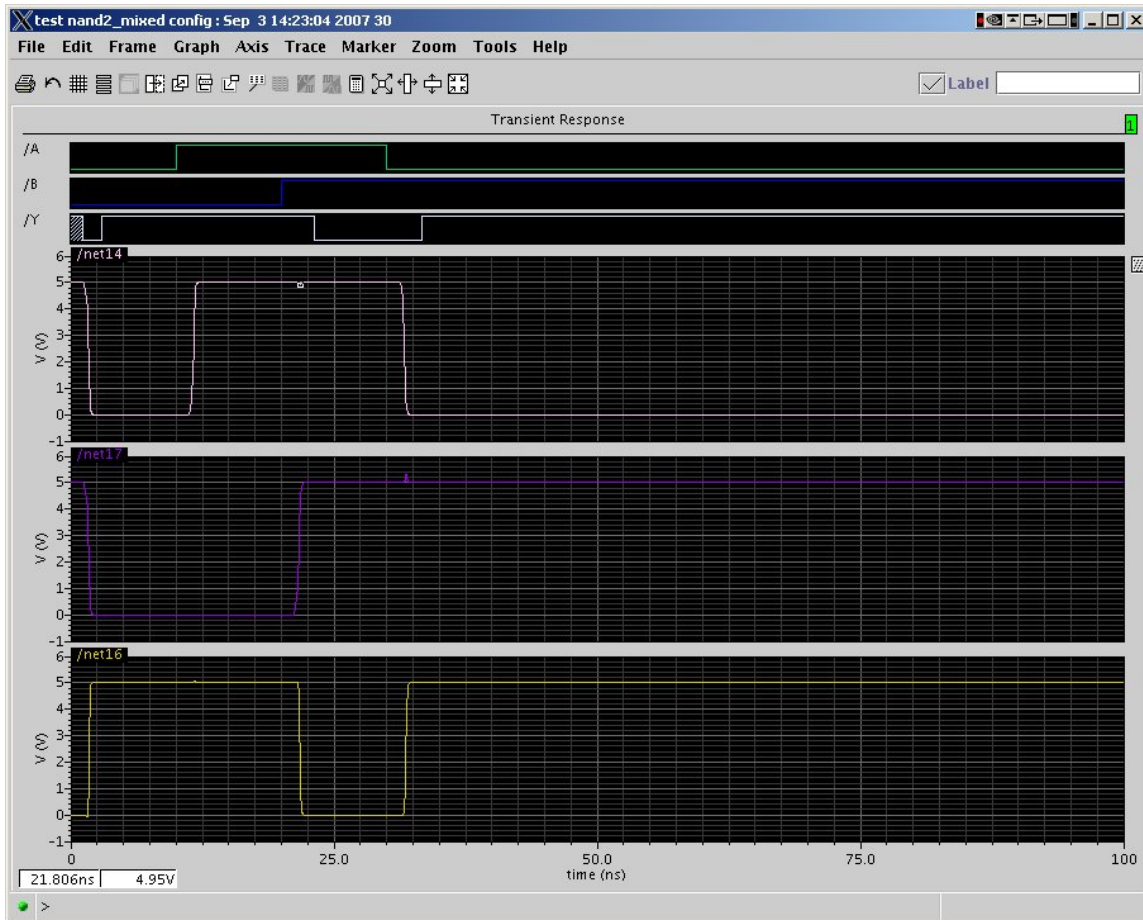


Figure 6.24: Results of the mixed-mode simulation of **mixed-test**

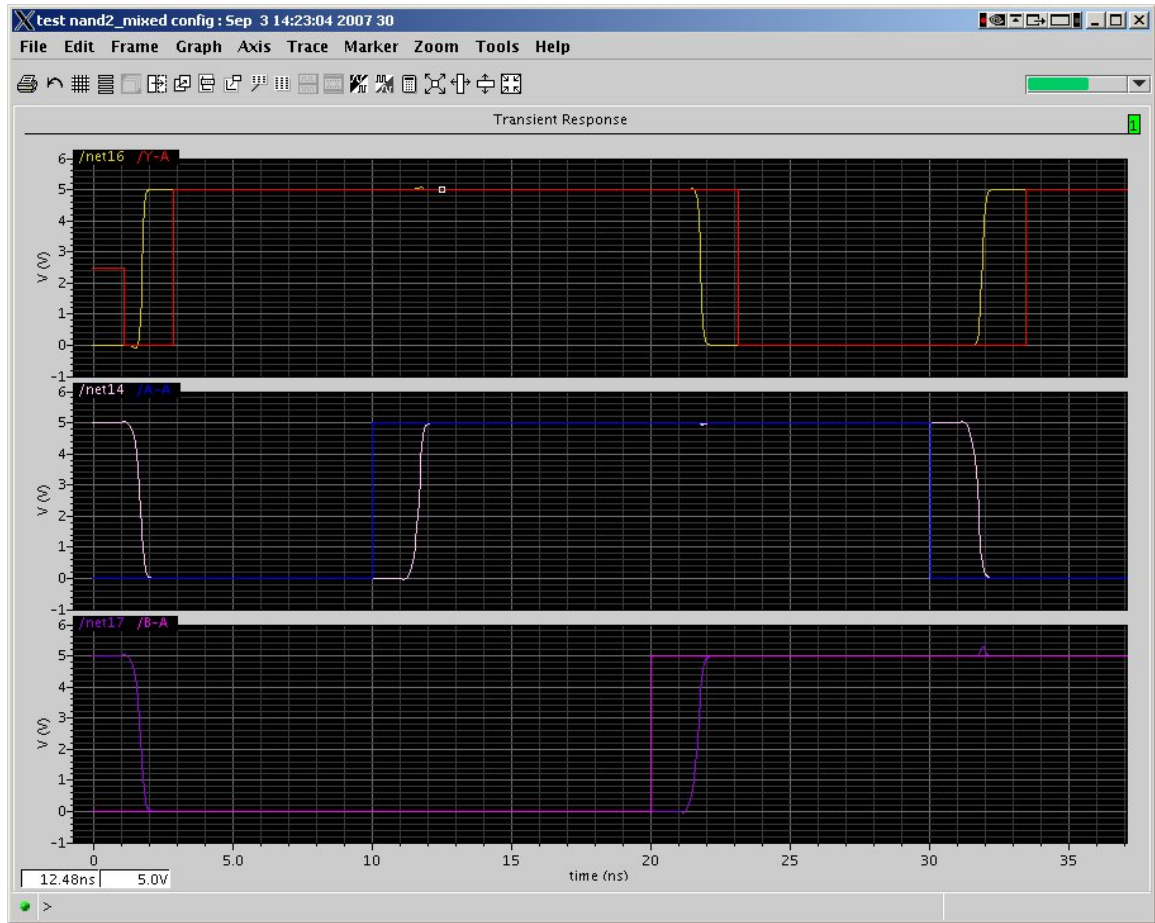


Figure 6.25: Rearranged results of the mixed-mode simulation

```
Switching from DC to transient.  VERILOG time 0 (units of 100ps) corresponds to spectre time 0.

Message!  At the end of DC initialization the logic values
          of the following ports are X (unknown):

          net16
          net18

                                          [Mixed_Sig]
          "IE.verimix", 4: ...
ab = 00, out = 1
ab = 10, out = 1
ab = 11, out = 0
ab = 01, out = 1
Verilog/spectre Interface: 165 messages sent, 167 messages received.
0 simulation events (use +profile or +listcounts option to count) + 29 accelerated events
CPU time: 0.0 secs to compile + 0.0 secs to link + 3.6 secs in simulation
End of Tool:  VERILOG-XL      05.81.001-p   Aug 23, 2006  10:58:39
```

Figure 6.26: `$display` output from the **mixed-test** simulation

go? They went into a very distant file. Specifically, they went into:

```
<your-cadence-dir>/simulation/<schName>/spectreVerilog/config/netlist/digital/verilog.log.
```

In this directory path `<your-cadence-dir>` is the name of the directory from which you started **Cadence**, and `<schName>` is the name of the schematic that you were simulating (**mixed-test** in this case). If you look into that file you'll see your `$display` output near the bottom. For this example the output is seen in Figure 6.26.

That's not very convenient, but at least it is there if you need to look for it. You can use this to trap errors using **if** statements the same way you would in a normal Verilog-XL simulation, just make sure that you're testing digital values against digital values. If you try to look at an analog value in the Verilog testbench it's likely to be **Z** or **X**.

A more convenient solution is to specify a file for the outputs of the Verilog testbench. For example, a testbench for the **mixed-test** simulation could be written as in Figure 6.27. In this example I've opened a file for output using the `$fopen` Verilog command and then written my `$display` outputs to that file. The output is shown in Figure 6.28

Final Words about Mixed Mode Simulation

Note that mixed-mode simulation is a little touchy and delicate. There are lots of ways to break it. But, when it works it's a great way to simulate larger circuits with **Spectre** and still provide digital input streams using a Verilog testbench. It is the recommended way to drive complex circuits so that you can write testbenches in Verilog, and you can also write self-checking analog simulations by including **if** statements in your testbench that check

```
// Vermix stimulus file.
// Default verimix stimulus.

integer file; // declare the file descriptor first
initial
begin
    file = $fopen("/home/elb/IC_CAD/cadencetest/testout.txt");
    a = 1'b0;
    b = 1'b0;

    $fdisplay(file, "Starting mixed-test simulation of NAND");
    $fdisplay(file, "using digital inputs to an analog simulation");

    #10 $fdisplay(file, "ab = %b%b, out = %b", a, b, dout);
    if (dout != 1) $fdisplay(file, "Error - that's wrong!");

    a=1;
    #10 $fdisplay(file, "ab = %b%b, out = %b", a, b, dout);
    if (dout != 1) $fdisplay(file, "Error - that's wrong!");

    b=1;
    #10 $fdisplay(file, "ab = %b%b, out = %b", a, b, dout);
    if (dout != 0) $fdisplay(file, "Error - that's wrong!");

    a=0;
    #10 $fdisplay(file, "ab = %b%b, out = %b", a, b, dout);
    if (dout != 1) $fdisplay(file, "Error - that's wrong!");

end
```

Figure 6.27: **mixed-test** testbench file with file I/O

```
Starting mixed-test simulation of NAND
using digital inputs to an analog simulation
ab = 00, out = 1
ab = 10, out = 1
ab = 11, out = 0
ab = 01, out = 1
```

Figure 6.28: **mixed-test** testbench file with file I/O

the digital versions of your outputs.

6.5 DC Simulation

The Spectre simulations that have been described in this Chapter until now have been transient analysis. That is, you are applying a time-varying signal to the circuit and you're getting a simulation of the time-varying output of the circuit in response to that input. This is valuable stuff, and is generally what digital designers are mostly concerned about since this relates to the operating behavior of the digital circuit over time. However, when we're looking at the behavior of the devices themselves, we may want to do a *DC analysis* that looks at the steady state instead of the time-varying state. For example, Figure 2.7 in your text [1] is a plot of steady state behavior of an nmos transistor mapping the I_{ds} current as V_{ds} is changed. In fact, there are five different DC analyses in this figure. Each of the five curves is a DC analysis that holds the V_{gs} at a fixed level, then sweeps the V_{ds} from 0v to 5v and plots the resulting I_{ds} .

To run the same DC analysis in Spectre, start by creating a schematic with a single **nmos** device and two **vdc** supply nodes from either the **NCSU_Analog_Parts** or the **UofU_Analog_Parts** libraries. The Composer version looks like Figure 6.29. To perform a DC analysis on this circuit you would fix the gate voltage (it's fixed at 3v in this Figure 6.29), and then do a DC sweep of the V_{ds} by sweeping the voltage of the other **vdc** component from 0 to 5 volts. To see the I_{ds} value you would plot the current at the transistor's drain instead of the voltage of a node.

Now you can follow the following steps to complete the DC analysis of this circuit:

1. Open the schematic in **Composer**. Set the voltages of the **vdc** components as they are in Figure 6.29.
2. Select **Tools** → **Analog Environment**. When you choose the **analysis type** select **DC** instead of **tran**. This changes the **Choosing Analysis** dialog box to **DC**.
3. Choose your sweep variable for the DC analysis. In this case we're going to sweep the voltage of the **vdc** component that's connected to the **nmos** drain so choose **Component Parameter** so that we can select the component that we want to use for the sweep.
4. This changes the dialog box so that you can tell it which component you're going to select, which parameter of the component you want to vary, and what the sweep range is of that parameter. Click on **Select**

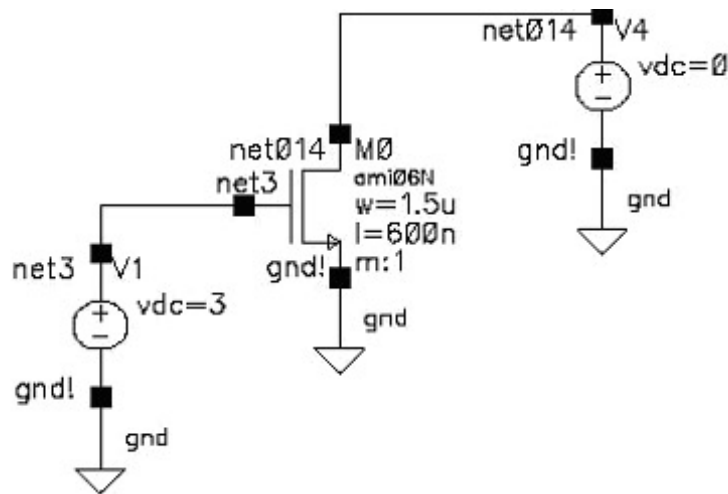


Figure 6.29: Simple circuit for DC analysis (**schematic** view)

Component so that you can select the **vdc** connected to the transistor drain by clicking on it in the schematic.

5. When you select (click on) the **vdc** connected to the transistor drain, you get another box (Figure 6.30) that asks you to select the component parameter that you want to sweep. We want to sweep the **dc voltage**. Click on that and then OK. You'll see that the component name (**V1** in my circuit) and the parameter name (**dc**) have been added to the **Choosing Analysis** dialog box which now looks like that in Figure 6.31.
6. Now enter the Sweep Range. Start from 0 and stop at 5. This will set up the DC analysis to sweep V_{ds} from 0v to 5v during the analysis. Click OK.
7. Back in the Analog Environment you need to select the outputs to be plotted. Use the same **Select on Schematic** choice that you've used before, but instead of selecting a wire, select the red square on the drain of the **nmos** transistor (the top leg of transistor M0 in Figure 6.29 - the one without the arrow). This will cause a circle to be put around the drain, and the drain current will be plotted. This shows up as **M0/D** for the output in my circuit (**M0** is the transistor identifier, and **D** is the drain current). The **Analog Environment** dialog box should look like Figure 6.32 when you're done.
8. Now run the simulation. You should get a waveform output like Figure 6.33 that has a single I_{ds} curve for a 3v gate voltage (V_{gs}) and a DC sweep from 0v to 5v on V_{ds} .

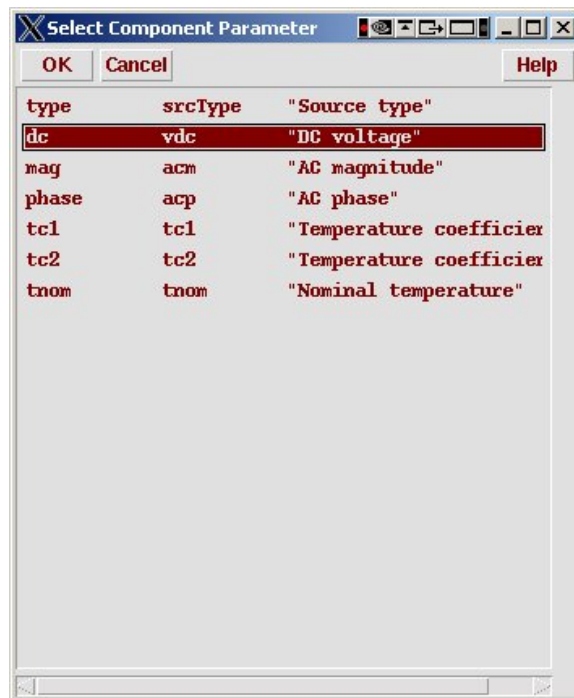


Figure 6.30: Component parameter selection dialog box for DC analysis

6.5.1 Parametric Simulation

To generate a graph like Figure 2.7 in the book, you need to run five DC analyses and plot them all together. Fortunately, there is an easy way to run all five in Spectre. It's called *parametric simulation*. To get a parametric simulation of all five curves in Figure 2.7, you need to run five different simulations, each one with a different V_{gs} . In the book they are simulating a 180nm process with a 1.8v power supply. We'll be using our 0.6μ (600nm) process with a 5v supply, so we'll sweep from 1v to 5v ($V_{gs} = 1v, 2v, 3v, 4v$ and 5v). Here's how to set up that parametric simulation:

1. Quit the Analog Environment and go back to editing your schematic.
2. Edit the properties of the **vdc** component connected to the gate of the transistor (use **Q** to get the properties dialog box). Change the DC Voltage value from **3 V** to **foo V**. What you've done is change the fixed value of 3 into a variable named **foo**.
3. Start up the Analog Environment and set up the same DC analysis as before (DC analysis with **Component Parameter** as the **Sweep**

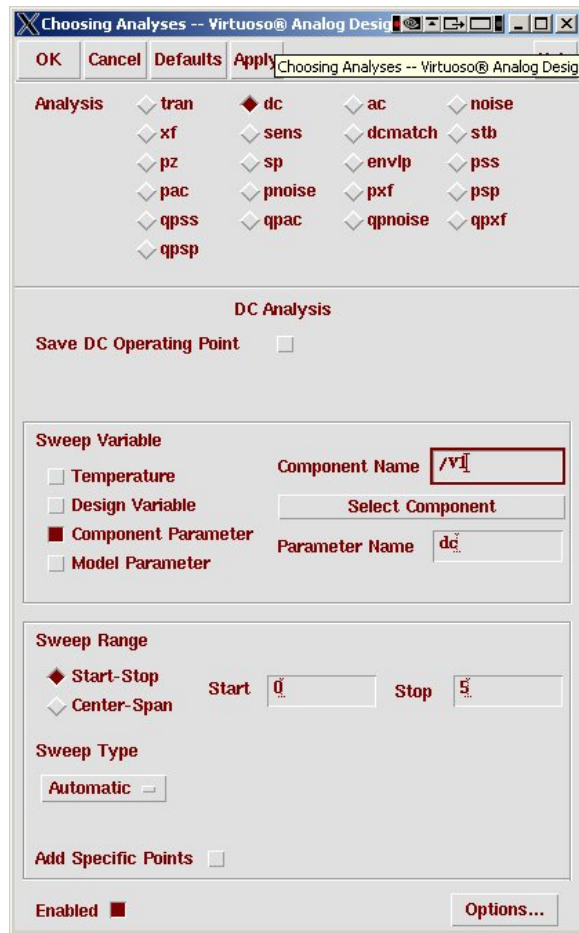


Figure 6.31: DC analysis dialog box

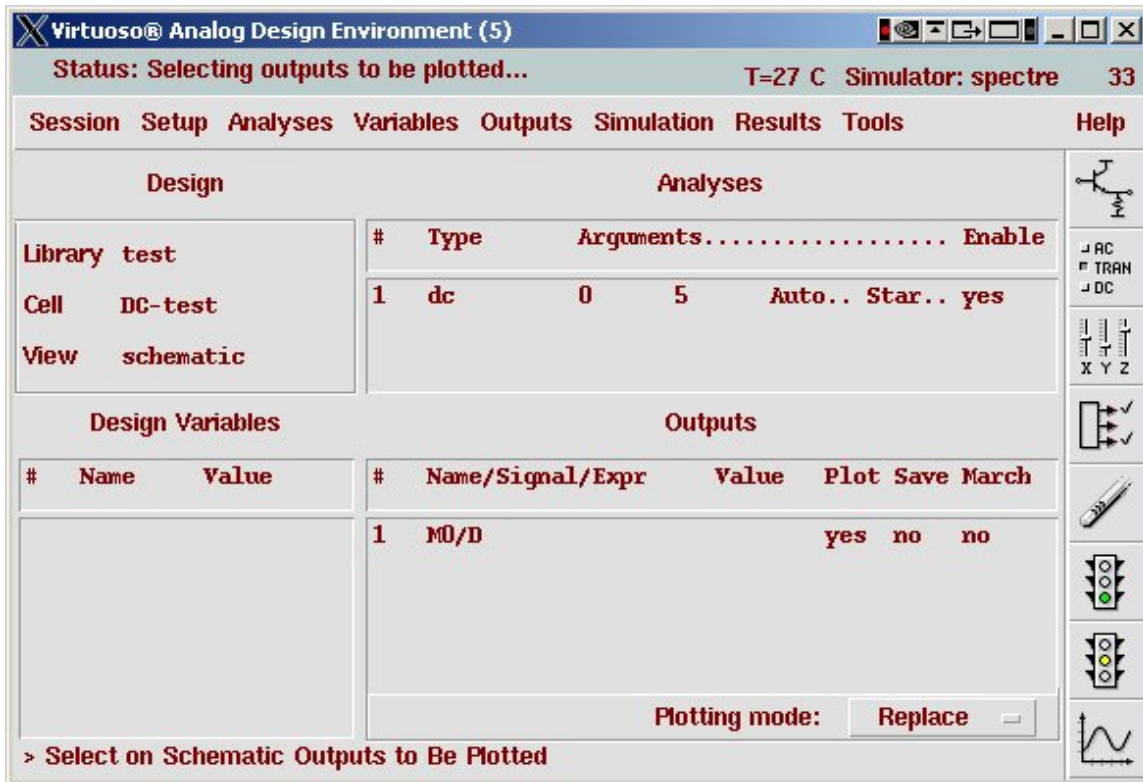


Figure 6.32: Analog Environment dialog box for DC analysis

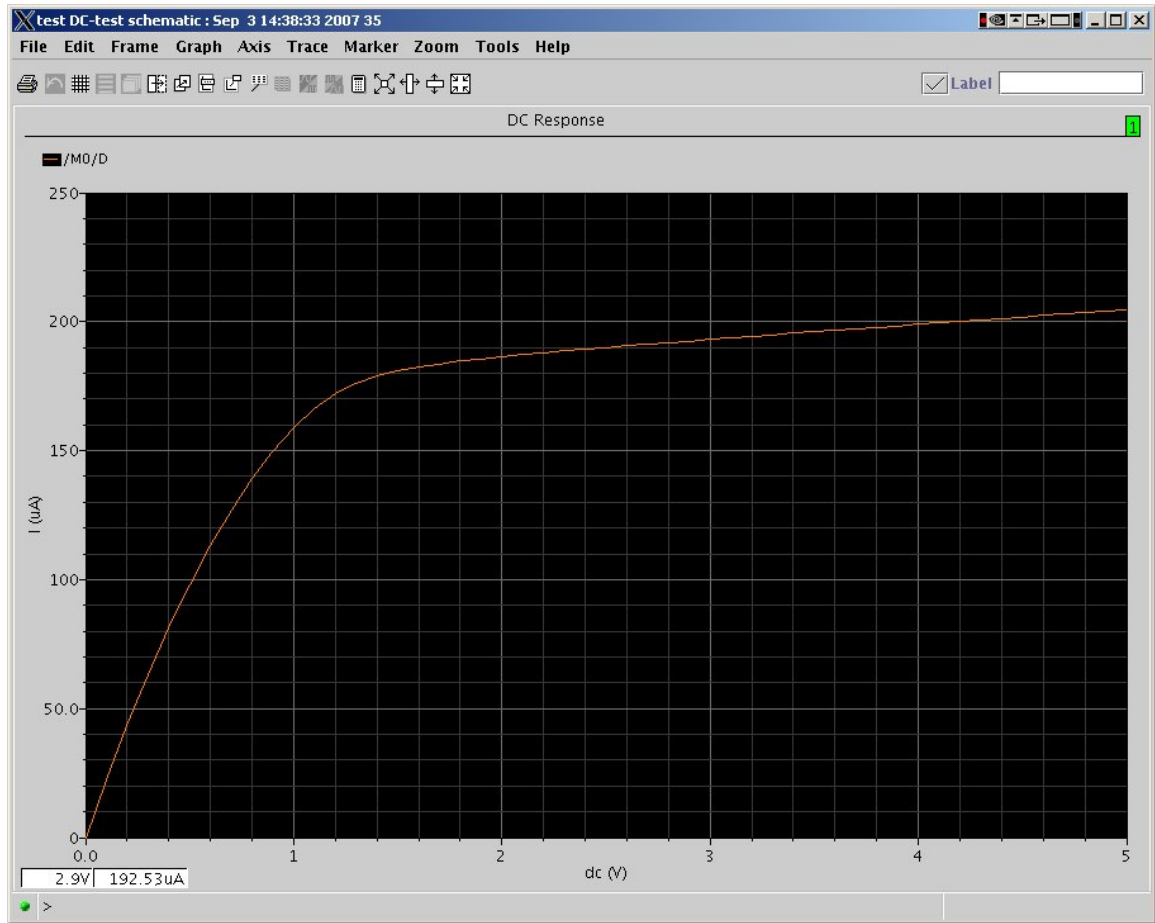


Figure 6.33: DC analysis output waveform for a single set of parameters

- Variable**, the **vdc** connected to the drain as the component, and **dc voltage** as the parameter name. Sweep the voltage from 0 to 5 volts).
4. Select the drain current as the output to be plotted, the same as before. For parametric simulation it's important that the selected outputs to be plotted show up in the **Analog Environment** control window as both **plot** and **save** so that the waveform viewer can see the results from all the runs. If **save** is not set you can set it by double-clicking the output name in the **Outputs** pane of the window.
 5. Now comes the tricky part. You have the V_{gs} voltage defined as a variable **foo**. You need to set that variable to some value. You could set it to a fixed value by selecting **Variables** → **Copy From Cellview** to get **foo** into the **Design Variables** pane, and then click on that design variable to set the value to whatever you want it to be (3 for example). Or, you can set things up so that all five DC analyses are run with **foo** set to a different voltage on each run.
 6. First select **Variables** → **Copy From Cellview** to get the **foo** variable into the **Design Variables** pane in the **Analog Environment** dialog box.
 7. Now select **Tools** → **Parametric Analysis**. This pops up a **Parametric Analysis** window.
 8. In this window enter **foo** in the **Variable Name** slot, and range the value of the variable from 1 to 5 with 5 total steps. The dialog box looks like that in Figure 6.34.
 9. Now choose **Analysis** → **Start** and all five simulations will run, and the results will be plotted in a single waveform window. The result looks like Figure 6.35. Note that the parameters that were used on each run (the value of **foo** in this case) are shown in the legend at the top of the graph.

This parametric analysis feature is very slick. You can name all sorts of parameters as variables in your schematics, and use the parametric analysis feature to simulate over a range of values. In this case we did five DC analyses: each one had a different fixed V_{gs} and swept the V_{ds} from 0v to 5v and plotted the resulting drain current of the nmos device. But, you can imagine using this technique to run all sorts of simulations where you want to vary parameters over multiple simulation runs. The basic technique of setting parameters in the components using variables and then using parametric analysis to do multiple simulation runs while varying the values of those variables is the same.

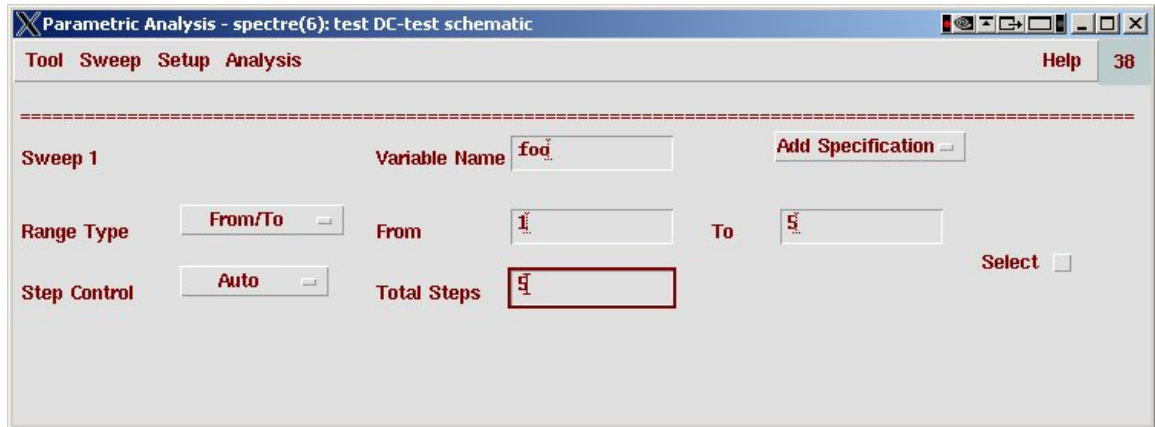


Figure 6.34: Dialog to set variable parameters for parametric simulation

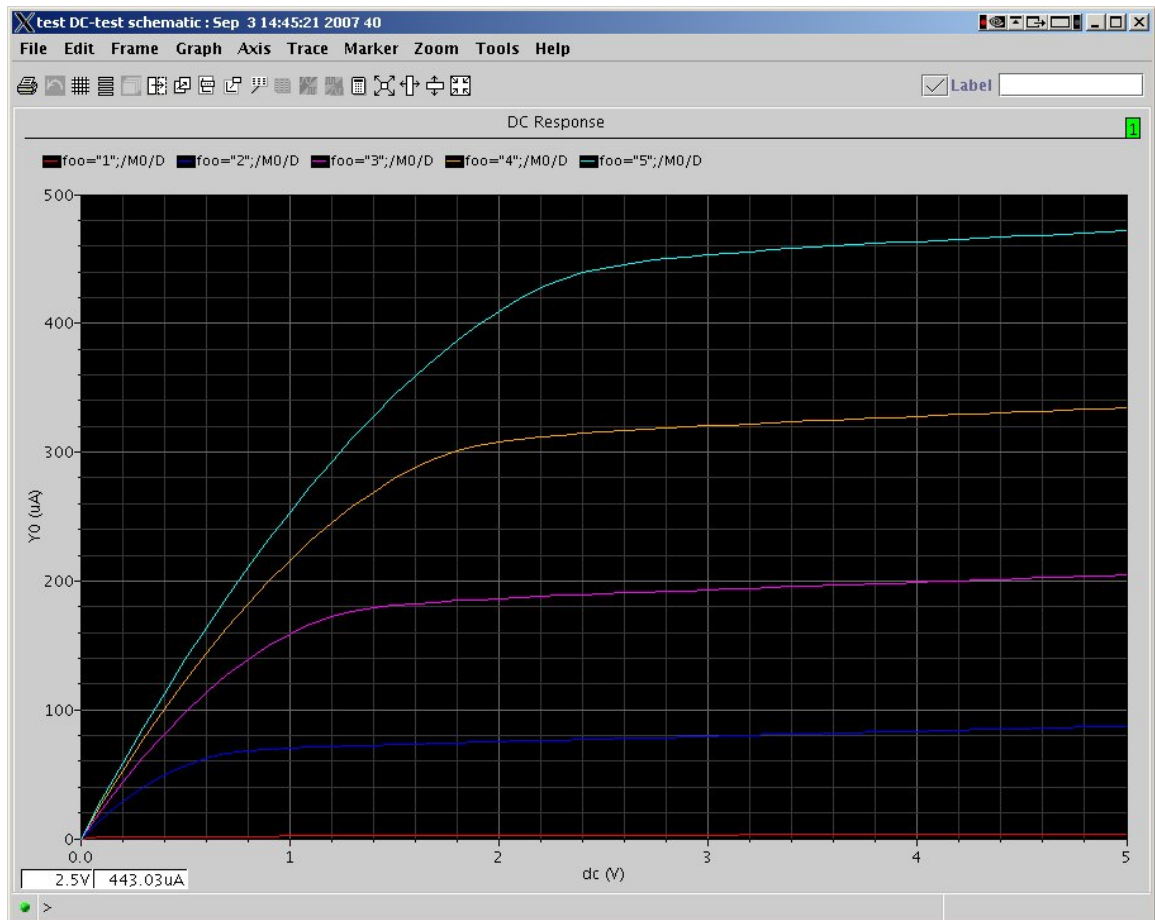


Figure 6.35: Output of parametric DC simulation with five curves

In particular, you can use the parametric simulation for **transient** simulation as well as for **DC** simulation. You could, for example, put a variable in the **capacitance** value of a load capacitor and do multiple transient simulations with varying amounts of output load capacitance. Or vary the width of transistors and run multiple transient simulations with different transistor sizes. You can vary just about any parameter that can be set in the **properties** box. It's a very powerful technique.