Chapter 3

Composer Schematic Capture

COMPOSER is the schematic capture tool that is bundled with the the dfII (Design Framework II) tool set. It is a full-featued schematic capture tool that we'll use for designing transistor level schematics for small cells, gate level schematics for larger circuits, and schematics containing a mix of gates and Verilog code for more complex circuits. In that case some of the components in the schematic will contain transistors at the lowest level, and some will contain Verilog code. Because the simulators that are used in conjunction with Composer are all Verilog simulators, these mixed schematics can be simulated using the same simulators used by schematics with only gates or transistors.

Although the schematic tool is called Composer in the documentation, it's called Virtuoso Schematic Editing in the window title. Virtuoso is also the specific name given to the layout editor in dfII. They're both part of the dfII Virtuoso tool suite I guess.

I find schematics extremely useful for all levels of design. Even for designs that are done completely in Verilog code I find that connecting the Verilog components in a schematic often makes things easier to understand than large pieces of code where connections are made with large argument lists and named wires. Your mileage may vary, of course.

Composer has connections to all sorts of other tools in the dfII tool suite, and to other tool suites. We'll look at all of them in future chapters.

Composer is a part of the IC v5.1.41 tools

- Composer is integrated with the Verilog-XL and NC_Verilog simulators so that you can automatically export a schematic to a simulator. The Composer/Verilog integration will take your schematic and generate a Verilog netlist for simulation, and also build a simple testbench wrapper as a Verilog file that you can modify with your own testing commands. We'll see how that works in the chapter on Verilog simulation.

A file that captures the component and connection information for a circuit is called a "netlist," and the process of generating that file is called "netlisting."

- There is also an interface that can take a schematic and convert that schematic to the Verilog structural file for input to a tool that uses that type of input. Synopsys dc_shell for synthesis and Cadence SOC

Encounter for place and route are just two of the possible tools that the structural Verilog file can be used with. This interface is known as the Cadence Synopsys Interface, or CSI, but it is a general way to convert a schematic to a structural netlist.

- Composer has a connection with the Cadence Virtuoso-XL layout tool so that the designer can see the connection between the layout and the schematic. This can be used as a guide for producing layout based on a schematic using Virtuoso-XL, and is also a mechanism for specifying the connectivity of a circuit when using the ICC Chip Assembly Router to assemble large chip pieces.
- There's also a connection the Affirma Analog Environment which is an interface to the Cadence Spectre analog simulation tool. Using this interface you can, for example, pick circuit nodes from your schematic to be plotted in the analog simulation output file. This connection will also generate the required Spectre or Spice netlist from your schematic automatically.
- You can generate a schematic by hand, or you can generate a schematic automatically from a Verilog structural netlist. For example, you can take the output from Synopsys dc_shell or Cadence BuildGates synthesis and generate a schematic from that structural Verilog file. The generated will be a mess to look at! But, it will be very useful for using the Cadence LVS (Layout Versus Schematic) checks using either Diva or Assura LVS. This will compare a layout to a schematic to make sure they are structurally the same.

You could easily use gates from the NCSU_Digital_Parts library for this tutorial if you are using the CDK without local enhancements.

For now, this chapter will introduce Composer by drawing schematics for simple circuits using standard cell gates and modules from the UofU_Gates library, and transistors from the NCSU_Analog_Parts and UofU_Analog_Parts libraries. In order to follow these steps you must have started up Cadence icfb with the NCSU CDK extensions. This was explained in Chapter 2, and is done using the `cad-ncsu` script.

3.1 Starting Cadence and Making a new Working Library

Now that you have your own cadence directory (called ~/IC_CAD/cadence if you've followed the suggestions up to this point), remember to connect to that directory before starting up Cadence. Also make sure you have your LOCAL_CADSETUP environment variable set so that you get the class extentions to the Cadence setup. Directions for setting this up were described in Chapter 2.

1. Start up Cadence dfII by running the command `cad-ncsu`. as described in Chapter 2. You may have used (and may still be using) a different setup for a different class, but please use `cad-ncsu` for this class. You should get a window (called the **Command Information Window** or CIW) similar to the one shown in Figure 2.1.

2. You should also get another window for the **Library Manager** as shown in Figure 2.2. The libraries that you'll see in the default **Library Manager** window are described in Chapter 2.

3. In order to build your own schematics, you'll need to define your own library for your own circuits. To create a new working library in the library manager, select **File** → **New** → **Library**. In the **Create Library** window that appears fill in the Name field as `tutorial`, or whatever you'd like to call your library. Leave the **Path** field blank so that it creates the new library under the directory in which you started Cadence. If you want to create a library somewhere other than in your ~/IC_CAD/cadence directory you can put the whole path in the path field. In the **Technology Library** box select **Attach to an existing tech library** and choose the **UofU AMI 0.60u C5N** technology library. The dialog box is shown in Figure 3.1 Now press OK and the new library will show up in your Library Manager window.

Librarys are defined in a `cds.lib` *file that is created in your cadence directory.*

Don't make your library name start with a number, and don't use "-" or "." in the name! An underscore is all right.

Now the working library has been created. All the project cells (components) that you generate should end up in this library. When you start up the **Library Manager** to begin working on your circuits, make sure you select your own library to work in.

3.2 Creating a New Cell

When you create a new cell (component in the library), you actually create a view of the cell. For now we'll be creating "schematic" views, but eventually you'll have lots of different views of the same cell. For example, a "layout" view of the same cell will have the composite layout information in it. It's a different file, but it should represent the same circuit. More about that later. For now, we're creating a schematic view. To create a cell view, carry out the following steps

We'll eventually use "cmos_sch" views for individual leaf cells and "schematic" views for cells with hierarchy. For now we'll just make "schematic" views to keep things simple.

Creating the Schematic View of a Full Adder

1. Select the **tutorial** library you just created in the Library Manager.

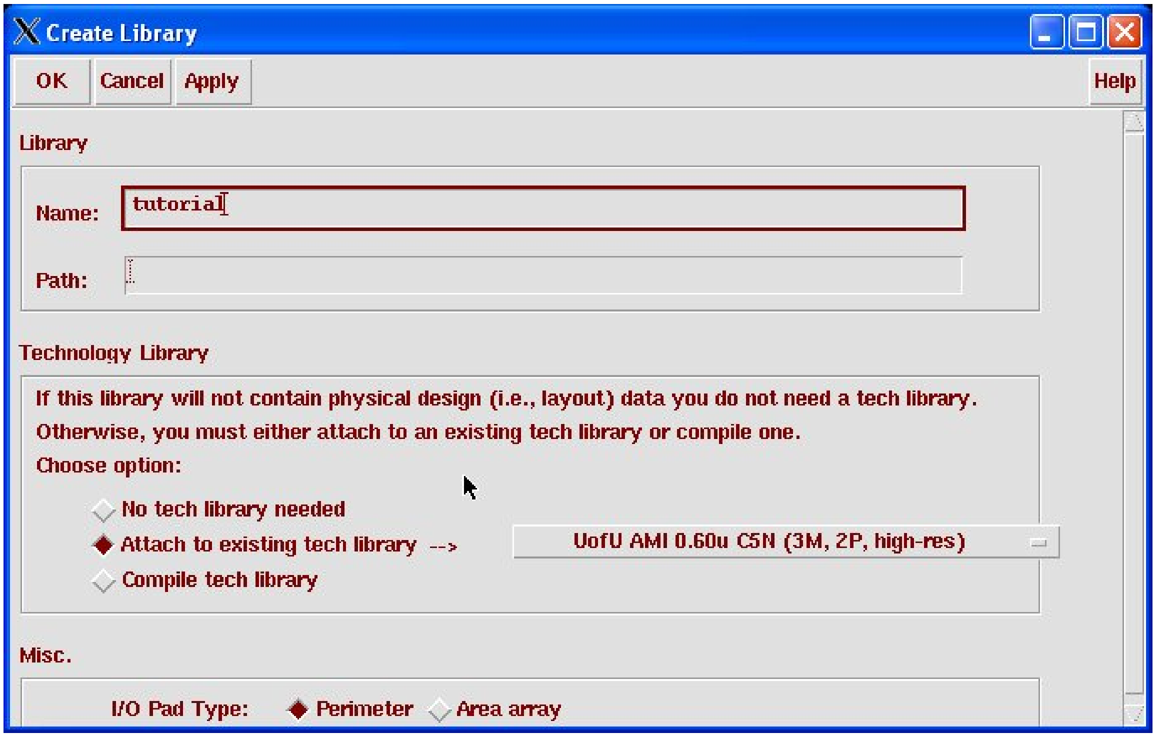

Figure 3.1: Library creation dialog box in Library Manager

2. Select **File** → **New** → **Cell View** from the **Library Manager** menu. The **Create New File** window appears. The **Library Name** field should be filled in as `tutorial` and the **Cell Name** field as `FullAdder`.

   Choose **Composer - Schematic** from the **Tool** list and the view name should be automatically filled as **Schematic**. This window is shown in Figure 3.2. Click OK.

3. A blank Composer schematic editing window appears which looks like that in Figure 3.3.

Many of the keyboard keys are bound to "hotkeys" which cause actions to happen. You can see these bindings in the menus.

4. Adding Instances: An instance (either a gate from the standard cell library, or a cell that you've designed earlier) can be placed in the schematic by selecting **Add** → **Instance** or pressing `i` on the keyboard, and the **Component Browser** dialog appears (shown in Figure 3.4). Choose a library from the **Library** list and then choose the cell from the list box and the outline of the component's symbol will appear when your curser is in the schematic window. Left clicks will add instances of that cell to your schematic.

   Additional options can be seen by popping up the **Add Instance** dialog box. In this box you can change, for example, change the orienta-

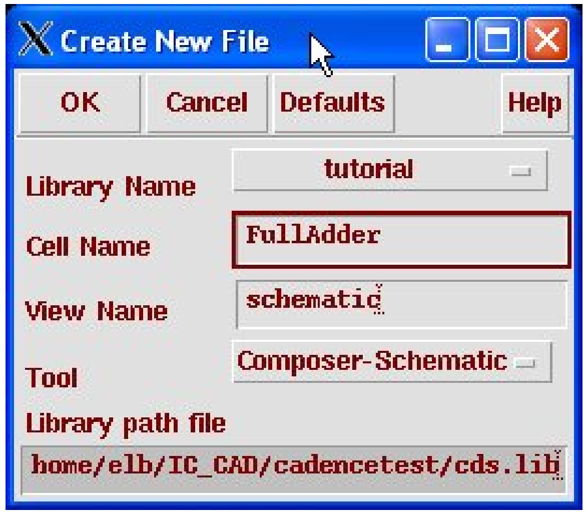

Figure 3.2: New cell dialog box in Library Manager

tion of the symbol by rotation or reflection which can come in handy for some schematics. This box is enabled with the `F3` function key. This box is shown in Figure 3.5.

*The **F3** function key often adds options to commands. Try it in different situations to see what it does.*

The instances you should add are "symbol views" of the instances. There are lots of "views" of cells that all have different features, but for schematics, symbols are what you want.

For example to add a 2-input NAND gate use the the **Component Browser** window, chose the library in the Library list as **UofU_Gates_v1_1** and cell in the list box as **nand2**. Alternately you can use the **Add Instance** window, fill the **Library** as `UofU_Gates_v1_1`, the **Cell** field as `nand2`, and the **View** field as `Symbol`. Still another way to add an instance is through the **Add** → **Instance** menu in the composer window.

Other instances can be added in the similar fashion as above. To come out of the instance command mode, press `Esc`. This is a good command to know about in general. Whenever you want to exit an editing mode that you're in, use `Esc`.

I sometimes just hit a bunch of `Esc`'s whenever I'm not doing something else just to make sure I'm not still in a strange mode from the last command.

5. Connecting Instances with Wires: To connect the different instances with wires select **Add** → **Wire (narrow)**, or press `w` to activate the wire command, or select the "narrow wire" widget from the toolbar on the left (it looks like a narrow wire . . .). Now go to a connection point of the gate instance and left-click on it to draw the wire and left-click on another connection point (another gate's input or output, or another wire, for example) to make the connection. If you would like

If you hover the mouse over the widgets on the left of the window you will get a little popup that tells you what that widget does.

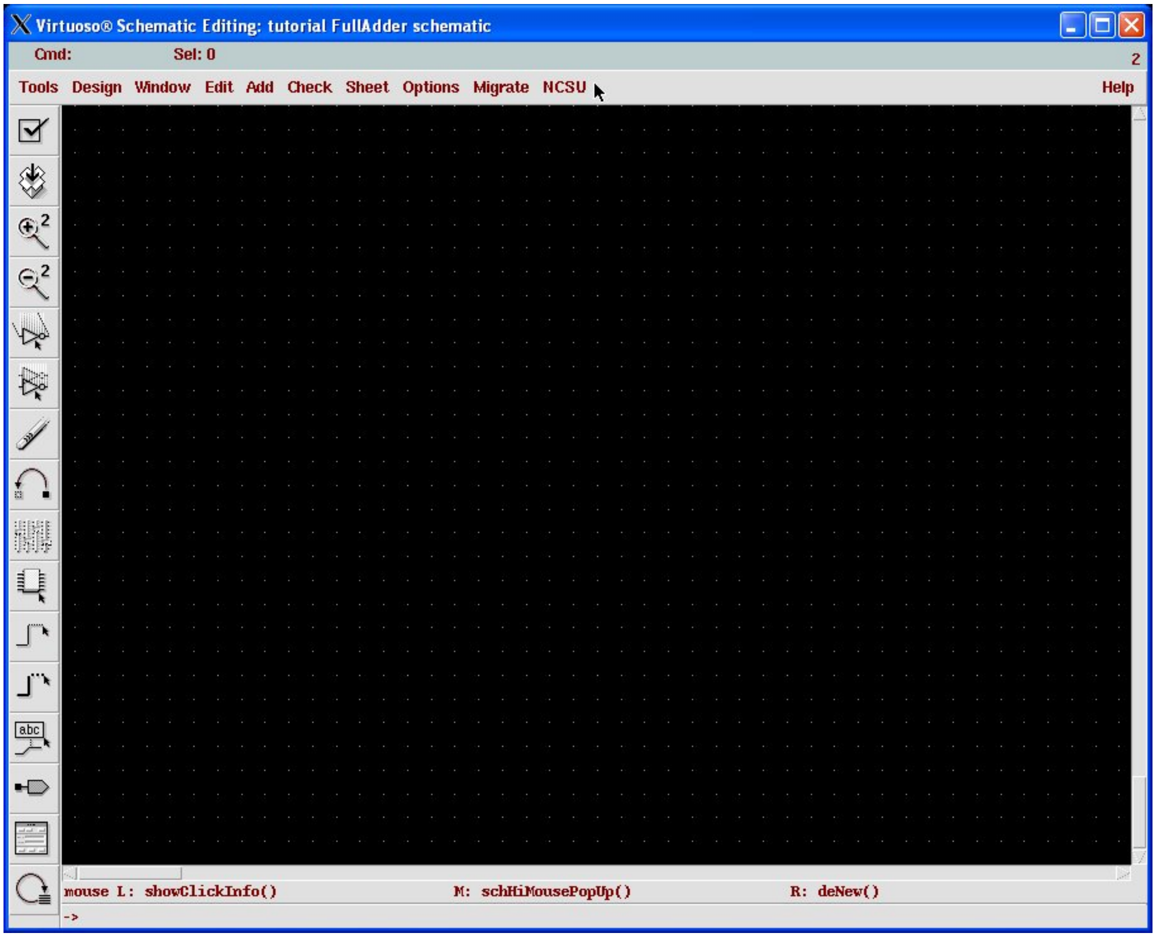

Figure 3.3: Blank Composer window - note command widgets on left of screen

to end the wire at any point other than a node (i.e. leave it hanging so that you can add a pin later on), double left-click at that point. To come out of the wire command mode, press `Esc`.

Note that because we're eventually going to simulate this with a Verilog simulator, the names you pick for pins and wires must be legal Verilog names. Verilog, for example, is case sensitive!

6. Adding Pins: Pins are connections that enter or leave this schematic sheet. They are called pins because they will correspond to pins on the symbol view of this schematic. Pins can be added by going to **Add** → **Pin** from the menu, or pressing `p`, or selecting the "Add Pin" widget from the left side of the Composer window to get the **Add Pin** dialog box (Figure 3.6). . For example, to put two input pins A and B, we can fill in the Pin Names field as A B (with a space) and the Direction list as input.

   Now go to the wire where you need to place the pin and left-click on

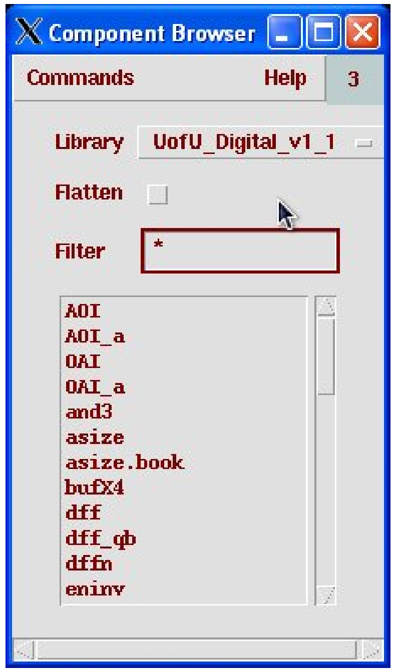

Figure 3.4: Component Browser dialog

it. The pin will be connected to the wire and look like a red pointer. If you need to rotate or flip the pins (i.e. to have an input coming in from the right instead of the left) use the buttons at the bottom of the **Add Pin** dialog box.

7. Other Command Functions

   Some common command modes and functions available under the Add and Edit menus in cadence are (of course, there are many more!):

   Under Add Menu:
   Add → **Wire (wide)** or press `W` to add a bus
   Add → **Wire name. . .** or press `l` to name wires
   Add → **Note** → **Note Text. . .** or press `L` to add a note

   Under Edit Menu:
   Edit → **Undo** or press `u`
   Edit → **Stretch** or press `s`
   Edit → **Copy** or press `c`
   Edit → **Move** or press `m`
   Edit → **Delete** or press `Delete` key

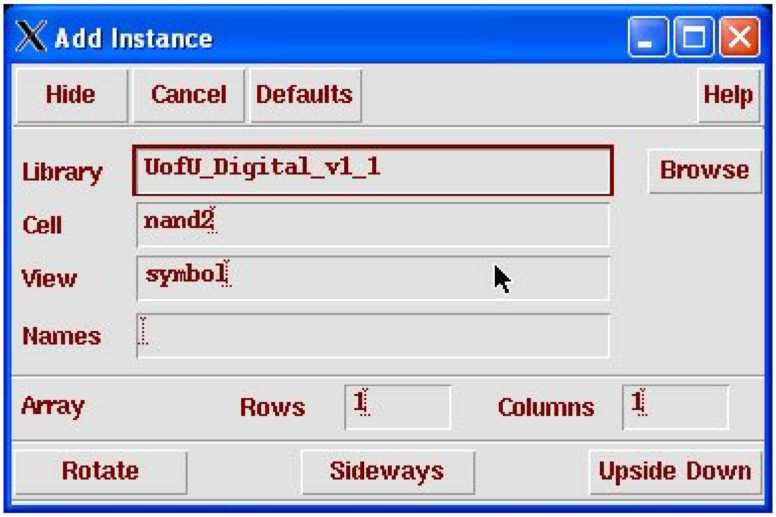

Figure 3.5: Add Instance dialog box

Edit → **Rotate** or press r
Edit → **Properties** → **Object** or press q

The **Edit Properties** command is a general command that can give you all sorts of information about whatever object you select. It's quite handy.

In general you can pick items in the schematic and move them using the left mouse button. You can usually select groups of objects by clicking and dragging with the left button, and you can zoom by clicking and dragging with the right button. All of these buttons are mode-dependent though. You can see the current bindings of the mouse buttons in the lower part of the Composer window.

8. It's good shematic practice to always put a border around your schematic. Borders can be found in the **UofU_Sheets** library. The asize sheet is a good one for small circuits because when you print the schematic all the gates and labels will still be visible. Larger sheets like the bsize, csize, etc. will cause the gates and labels to be too small to see when printed on 8.5 x 11 paper. When you add a border you can add your name and other info in the title block using the **Sheet** → **Edit Title ...** dialog box.

   Using the commands above to draw a full adder bit, and including an **Asize** sheet, and filling in the title block results in the schematic seen in Figure 3.7.

*Libraries sometimes group their cells into "categories." In this case you select the category first in the **Component Browser** before selecting the cell.*

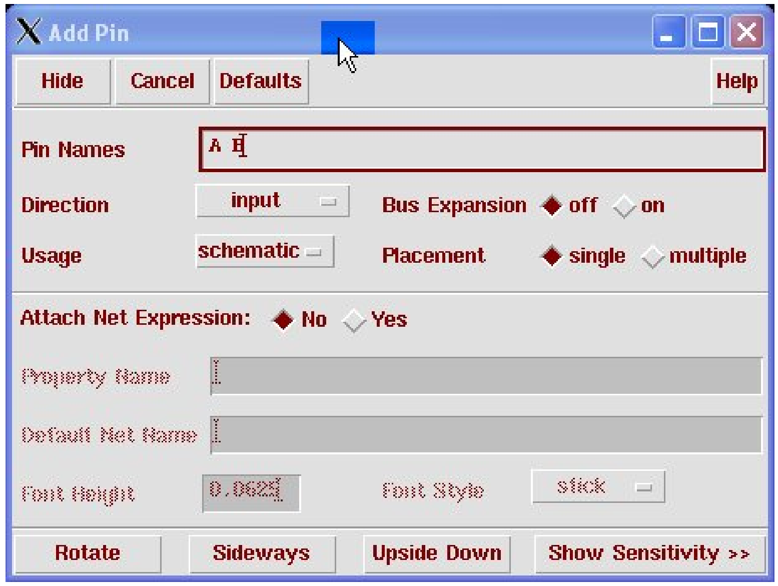

Figure 3.6: Add Pin dialog box

9. The design can be checked and saved by selecting **Design** → **Check and Save** or by pressing `F8`. For an error free schematic, you should get the following message in the CIW,

```
Extracting ''FullAdder schematic''
Schematic check completed with no errors.
''tutorial FullAdder schematic'' saved.
```

   Note that there's a big difference between the **Save** and the **Check and Save** commands. The **Save** command doesn't do any checks on the schematic! If you know there are errors that you haven't fixed yet but want to save so you don't lose work, use **Save**, but eventually you need to so **Check and Save** so that Cadence checks the schematic for errors. The CIW should not show any warnings or errors when you check and save. If it does, you should read and understand all of them before moving on. Some warnings may be ignored, but only if you're sure you understand what they are and that that are safe to ignore. For example, you can ignore the warning about outputs connected together if those outputs are coming from gates with tri-state outputs, but not if they are coming from regular static gates.

   After saving the design with no errors, you can close the window (and exit Composer if this was the last window) by selecting **Window** → **Close** or pressing `ctrl+w`. Or you can leave thee window open to go on to the next step.

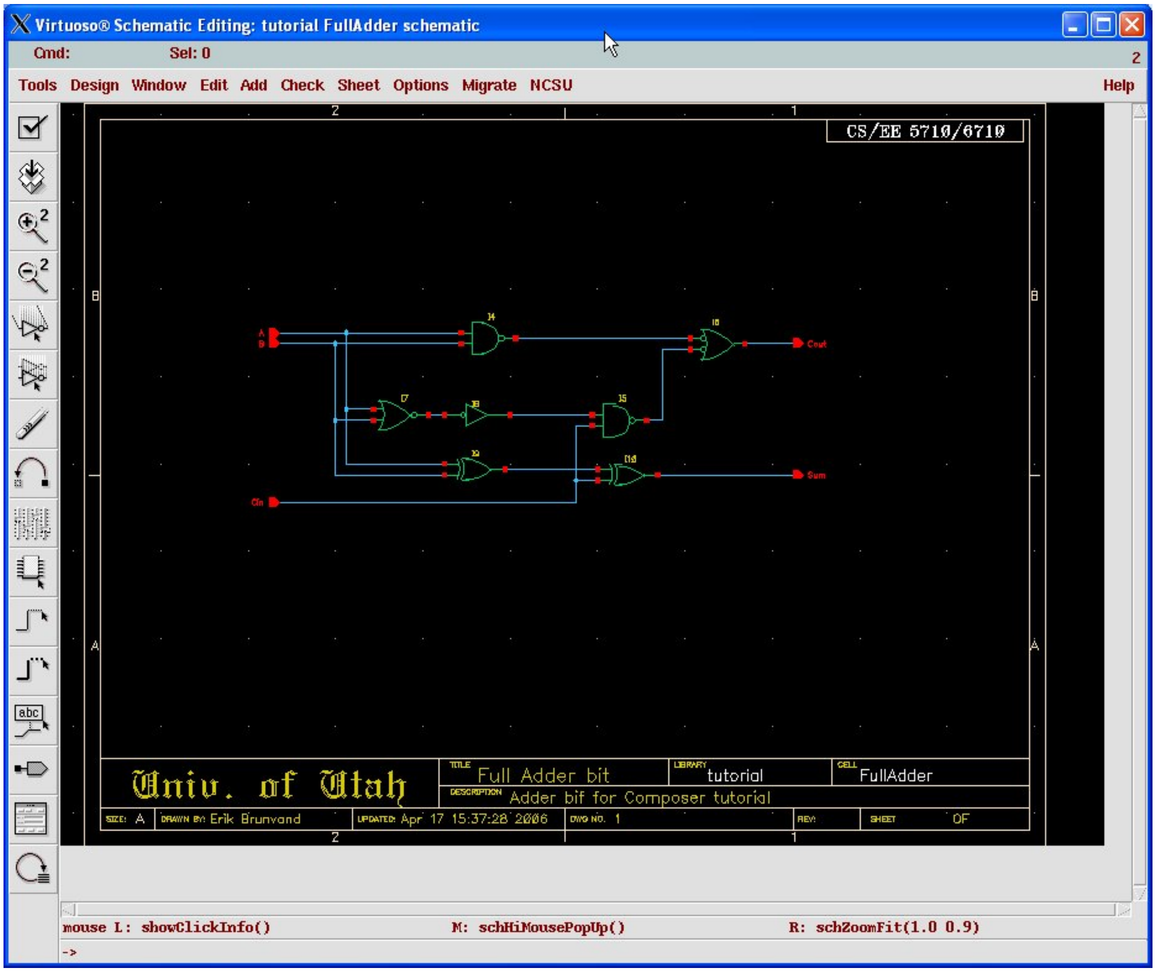

Figure 3.7: A Full Adder Bit Schematic

Creating the Symbol View of a Full Adder

You have now created a schematic view of your full adder. Now you need to create a symbol view if you want to use that circuit in a different schematic. A symbol is a graphical wrapper for the schematic so that it can be used to place an instance of that circuit in another circuit. The pins of the symbol must match one to one with the pins inside the schematic. The name must also be the same, but that will happen automatically if the schematic and symbol are different views of the same cell

1. In the Composer schematic window of the schematic you have created above, select **Design** → **Create Cell View** → **From Cell View**. A **Cell View from Cell View** window appears, press OK.

Figure 3.8: A simple symbol for the FullAdder circuit

2. In the **Symbol Editing** window which appears make modifications to make the symbol look as below. If you leave [@partname] in the symbol it will be filled in with the name of the cell when you instantiate this cell. If you want the symbol to say something different than the name of the cell you can replace [@partname] with some other text (**add**→**note**→**notetext**). The [@instanceName] placeholder will be filled in with an automatically generated instance number when you use the symbol. Note in the FullAdd schematic that there are yellow instance numbers above each gate. These are unique identifiers attached to every gate instance. An example of a very simple symbol for the FullAdd cell is shown in Figure 3.8.

   I haven't made many modifications to the automatically generated symbol in this case. All I've done is reorder the input and output pins in the symbol so that they'll connect more efficiently in a ripple-carry adder connection. You can format the symbol to make it look like the one in Figure 3.8 or use the edit commands to make the symbol look like whatever you like (use **add**→**shape** in the symbol editor, for example). Save the symbol and exit using **Window** → **Close**.

Once you have a symbol view, when you **Check and Save** the schematic it will also check that the pins in the schematic match up with the pins in the symbol. Now the full adder is ready to be used in other schematics.

Creating a 2-bit Adder using the FullAdder bit

Make sure that you have selected the **tutorial** library in the **Library Manager** and then select **File** → **New** → **Cell View** to make a new cell schematic. Fill in the **Cell Name** field as twoBitAdder and select the **Tool** from the list to be **Composer - Schematic**. This will make a new cell in the **tutorial** library for our new circuit.

In the Schematic Editing window which appears place two instances of the 1-bit full adder we created previously by selecting **Add** → **instance** or

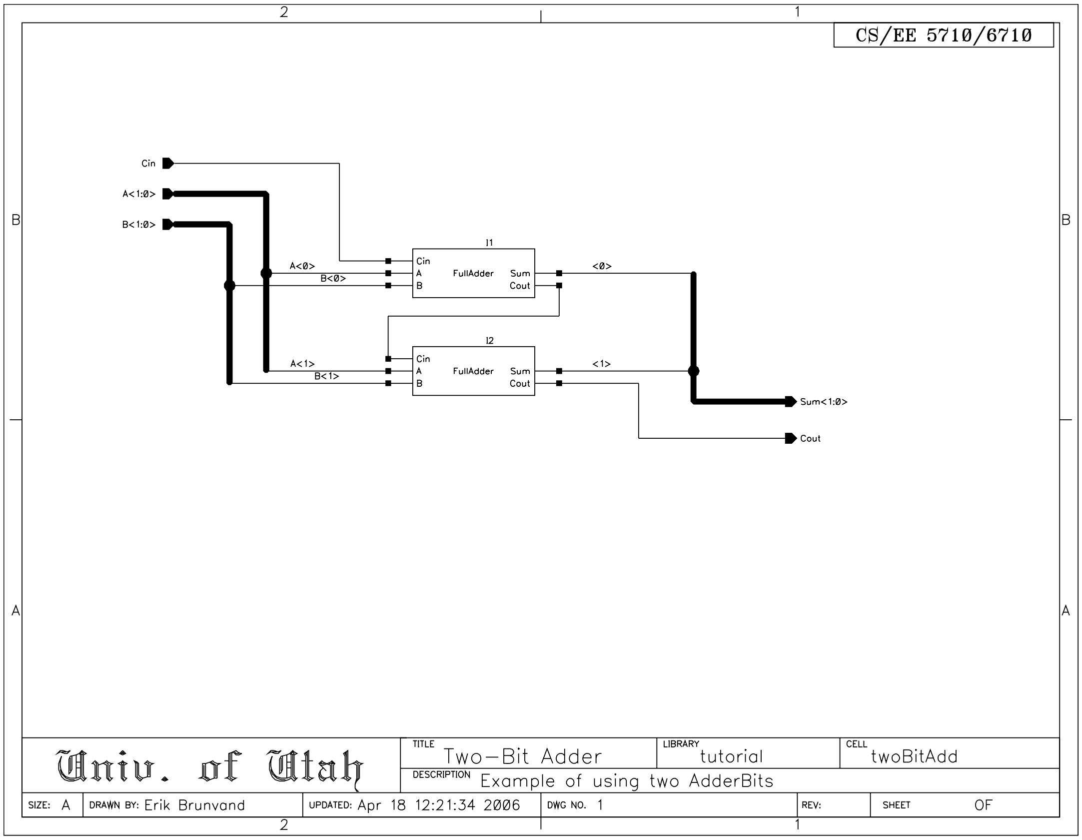

Figure 3.9: A two-bit adder using FullAdder bits

pressing `i` and attach them as in Figure 3.9. Remember to add an **Asize** sheet.

Note that the two-bit inputs and outputs have been bundled into two-bit buses in the shematic. Buses in Composer schematics are just wide wires, but they also have special naming conventions so that the tool knows how to assign names to each wire in the bus.

To add a bus in cadence, select **Add** → **Wire (wide)** or press `W` and then go about in a similar fashion as that of adding a wire to the schematic. The trickiest part of adding a bus is labeling. The pins connecting the buses are named in the following manner: A<1:0>, B<1:0>, where the numbers in the angle brackets represent the number of bits in the bus. The first number is the starting bit and the next number is the ending bit. Each wire in the bus must be named according to the expansion of this bus label. For example, in the A bus there are two wires named A<1> and A<0>. In Verilog

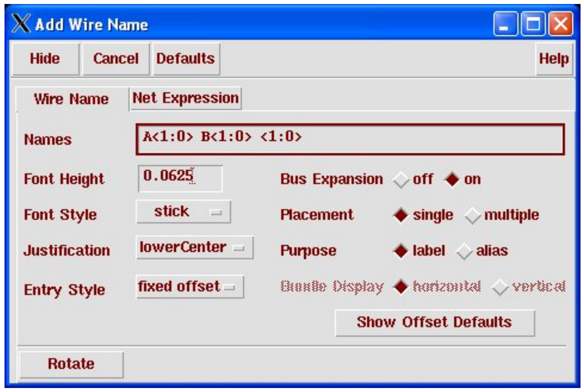

Figure 3.10: The Wire Name dialog box

buses are labeled in "little endian" mode which means that the high order bit should have the highest bit number and the low order bit the lowest. Thus, if you want the bus value to print out correctly later on your buses should be labeled A<1:0> and not A<0:1>.

In cadence the buses inherit the pin names and the nets tapping the buses inherit the bus name. This is why you can name the individual wires with the bit number only such as <0> and <1>. You can also name the individual wires A<1> and A<0> if you like. What is important is that every wire (thin single-bit wire or thick multi-bit wire) must be named so that the Composer knows which wires from the bus are being used. Wires (both thick and thin) can be named with the **Add** → **Wire Name . . .** , or `l` (after first selecting the wire to name), or the wire naming widget. The dialog box is shown in Figure 3.10.

The wire naming dialog box has **expand bus names** and **don't expand bus names** options that are useful for bus naming. If you give a bus-name like A<1:0> in the wire naming dialog box with the **expand bus names** checked, then each left click will name a single wire with one of the bus wire names (A<1> then A<0> for example). If the **don't expand bus names** option is used, then the next left click will name a (thick) wire with the whole name (A<1:0> in this example). Figure 3.11 has a closer view of the circuit so that you can see examples of how the bus naming works.

3.3 Schematics that use Transistors

Transistors can be used in designs just like any other "gate" primitives. They will eventually be simulated using the built-in transistor switch models in Verilog. They can also be simulated in an analog simulator like Spectre or

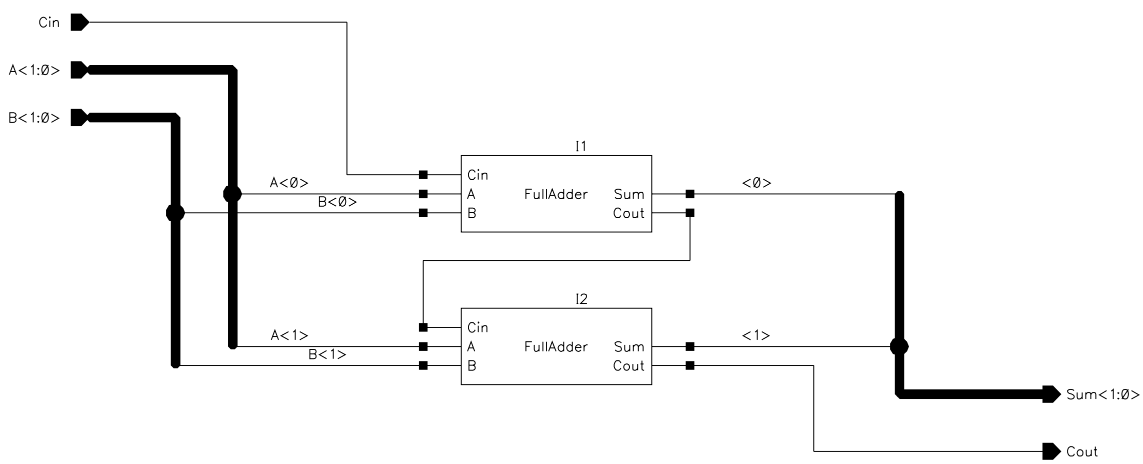

Figure 3.11: Close view of the twoBitAdd showing the bus naming

Spice. The generic transistor components are in the **NCSU_Analog_Parts** library. For now use the **nmos** and **pmos** transistors from those libraries. If you need four-terminal devices you can use **nmos4** and **pmos4** for devices with an explicit bulk node connection.

These will netlist into zero-delay switch models when they are simulated. If you'd rather have unit-delay simulation of your transistor switches you can use the same devices, but from the **UofU_Analog_Parts** library. The **UofU** library also has "weak' transistors that can be use for weak feedback circuits, and bidirectional transistors (the generic models must be used with their drain connections being the output connections - which is usually waht you want and speeds up simulation speed).

The following is an example of a simple NAND gate designed using transistors:

1. Select **File** → **New** → **Cell View** from the Library Manager menu or from the CIW menu. The **Create New File** window appears. The **Library Name** field should be `tutorial`. Fill in the **Cell Name** field to `nand2`. Choose **Composer - Schematic** from the Tool list and the view name is automatically filled as **Schematic**. Click OK.

   In the schematic window select **Add** → **Instance** or press `i`, and the **Add Instance** dialog appears. To add an NMOS transistor, in the **Component Browser** choose the library to be **NCSU_Analog_Parts**,

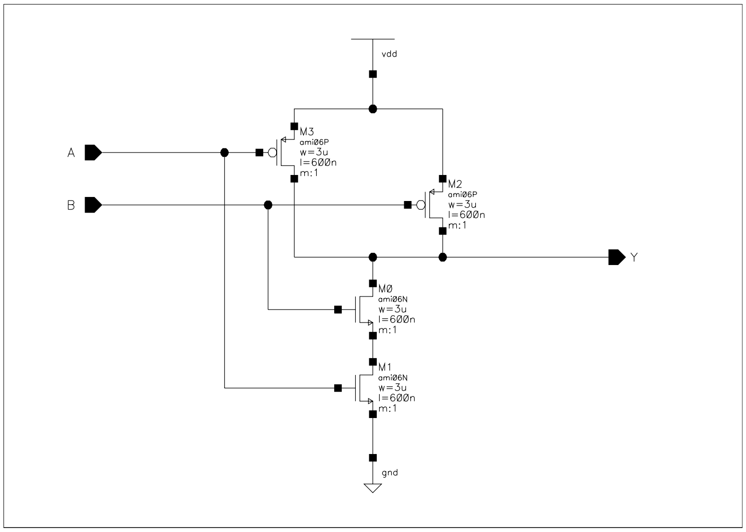

Figure 3.12: Transistor schematic for a two-input NAND gate

navigate through the **N_Transistors** category to the **nmos** cell. Or, directly in the **Add Instance** window fill the **Library** as `NCSU_Analog_Parts`, the Cell field as `nmos` and the View field as `symbol`. Add the PMOS transistor symbols to the schematic using the same procedure through the **P_Transistors** category to the **pmos** cell. Create the schematic of the NAND gate as in Figure 3.12 by adding wires and pins.

2. For transistor-based circuits you need to add the power supply connections explicitly. The vdd and gnd supply nets are in the **NCSU_Analog_Parts** library in the **Supply Nets** category. Connecting to the instances vdd and gnd in your schematic automatically connects these wires to logic 1 and logic 0 respectively in verilog simulations. We will connect those supply nets to particular voltages (+5v and 0v, for example) later in the analog simulations.

3. Transistors need a few more parameters than Boolean gates do. In particular, transistors need at least a length and width parameter to define their strength. You can do this in the **Add Instance** dialog box (you may have noticed that the **Add Instance** dialog was a lot bigger for the transistors than for the gates), or can do it after the fact by changing the properties of each transistor.

   Do change the properties select one of the nmos transistors and select **Edit** → **Properties** → **Object**, or press `q`, or use the **Properties** widget to bring up the properties window. The full properties window is shown in Figure 3.13.

Change the properties of the nmos transistor by changing the Width to `3u M` (3 microns). Leave the length as `600n M` (0.6 microns). Similarly follow the steps for the pmos transistor with W/L = 3/0.6 (i.e. W = `3u M` amd L = `600n M`).

4. Create a symbol for the NAND gate by selecting **Design** → **Create CellView** → **From CellView**. The Cadence-generated symbol will be a simple rectangle. You can easily modify it to make it look like Figure 3.14 using arcs and circles.

   Note that in order to get the circle for the nand2 bubble to be the right size in proportion to the rest of the gate you may have to use a finer grid while you're drawing that circle. You can change the grid size in the **Options** → **Display Options** dialog box, as the **Snap Spacing** value. But, when you're done make sure that you set the snap grid back to 0.0625 so that the pins you make will match up properly with the wires in the schematic!

3.4 Printing Schematics

Details of the .cdsplotinit file can be found in Chapter A

To print schematics you need to have printers set up by your tools administrator. The printers available to you are defined in a **.cdsplotinit** file. This file lives in the Cadence installation tree, but may also exist in a site-specific location for local printer information. It contains printer descriptions that describe which plot drivers should be used (usually postscript), and how to spool the output to the printer. There is usually at least one postscript or eps (encapsulated postscript) option defined so that if you plot to a file you can get a postscript version of your schematic.

To print (plot) a schematic, select the **Design** → **Plot** → **Submit...** menu choice. This will bring up the **Submit Plot** dialog box (seen in Figure 3.15). If all the choices are correct, you can select **OK** to plot the file. If you've selected a printer the schematic will print to that printer. If you've selected the **Plot To File** option you will get a file in the directory from which you started Cadence. Those are options that you have to select from the **Plot Options...** choice though.

When you click on the **Plot Options...** button you get another dialog box for the plot options as seen in Figure 3.16. This dialog box lets you set up all sorts of details about how you want your schematic plotted. The important options are:

Top Section: In this section you cand choose which plotter (printer) you wish to send your hard copy to with the **Plotter Name** selection. This

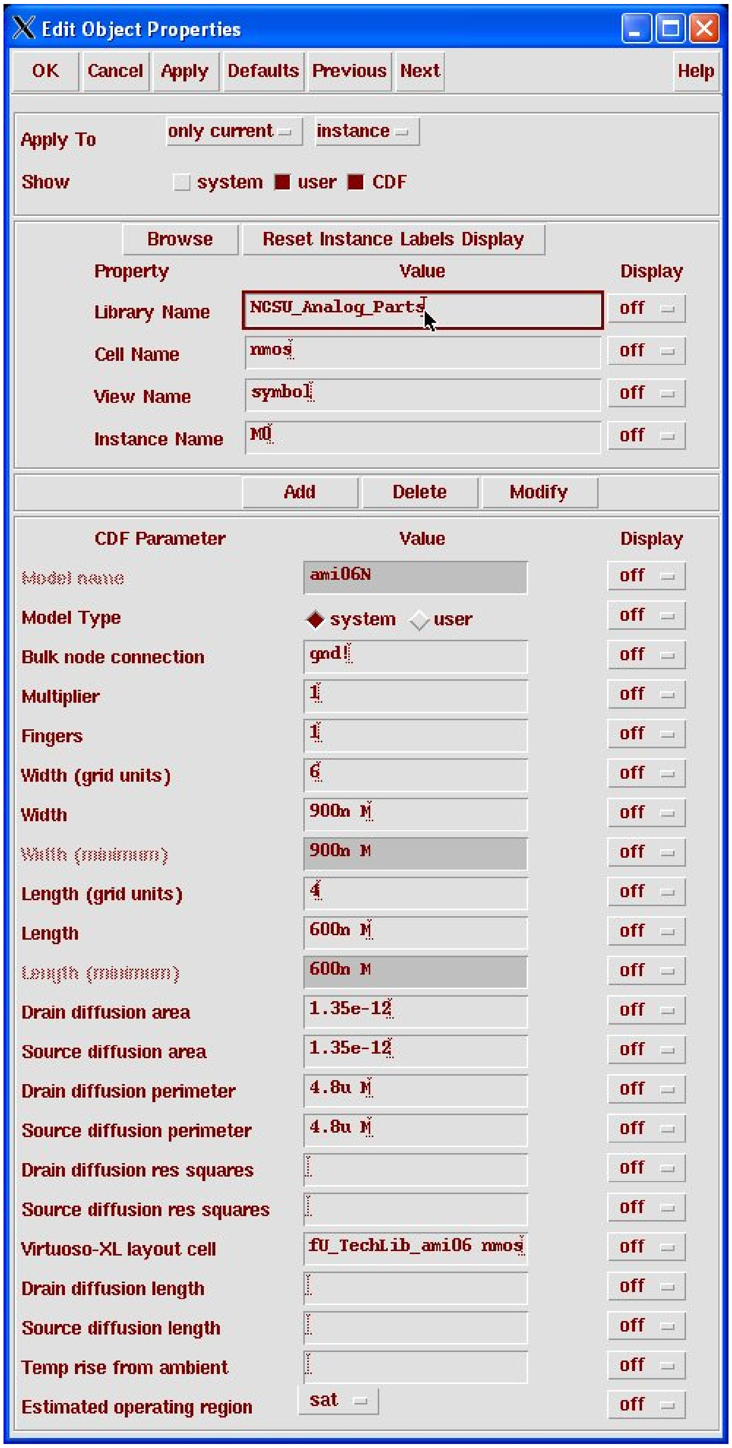

Figure 3.13: Device Properties for an **nmos** device

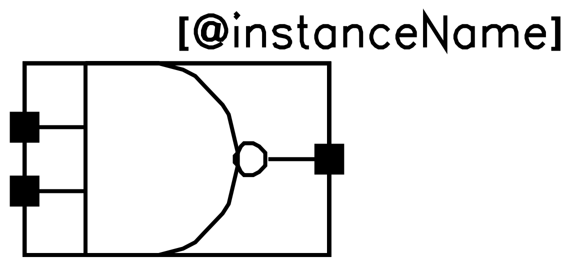

Figure 3.14: A symbol for the nand2 circuit

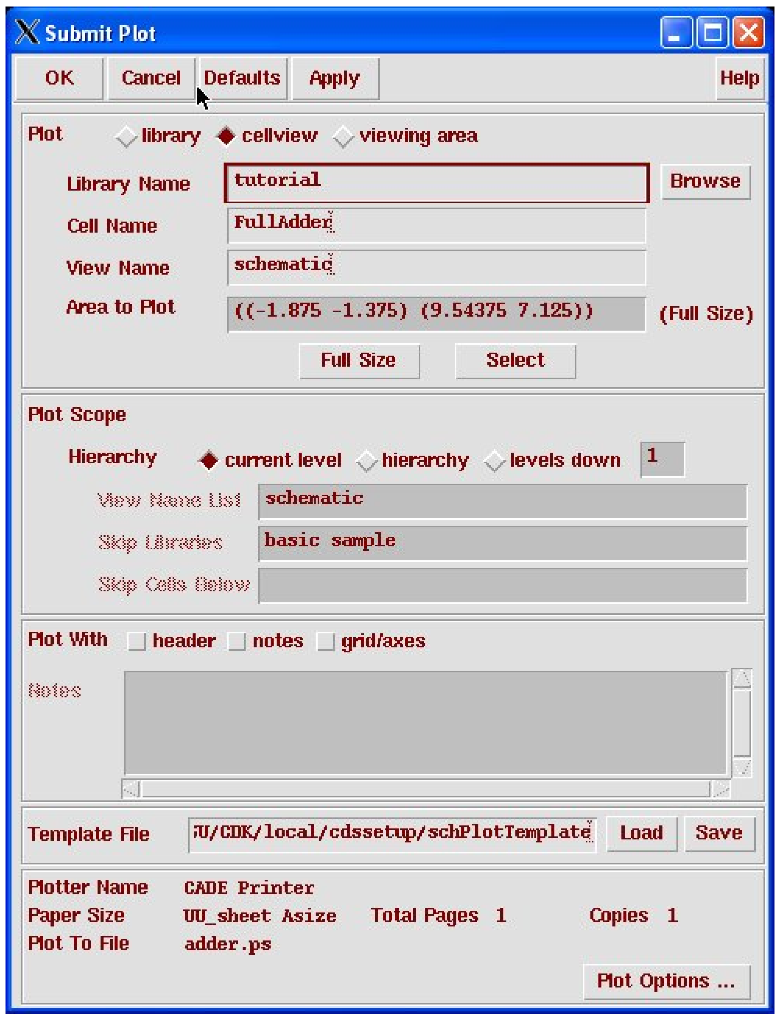

Figure 3.15: Dialog Box for Submit Plot

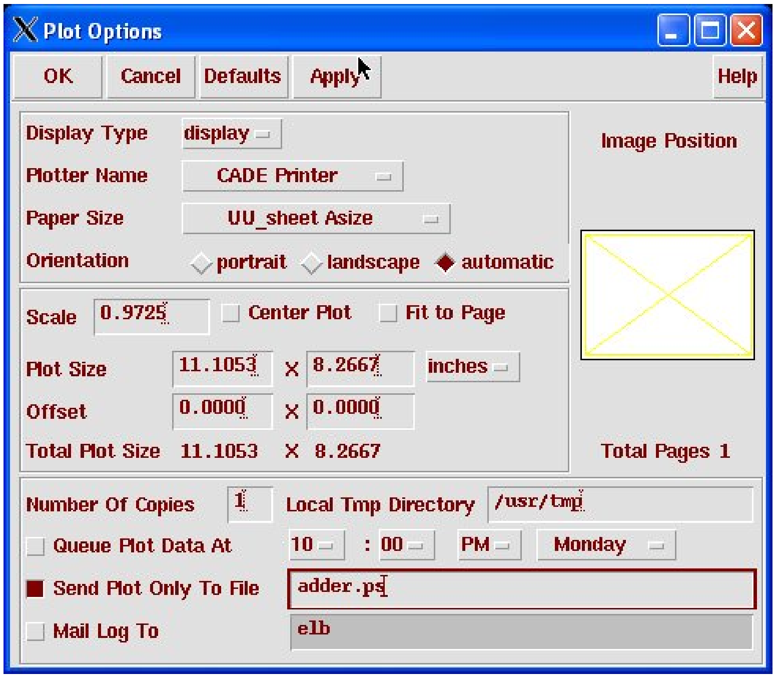

Figure 3.16: Dialog Box for Plot Options

also chooses a printer type for the **Plot to File** option. That is, if you want a postscript file, select a postscript plotter here. You can also choose a **Paper Size** if your printer has different sizes of paper loaded. Normally you use some sort of Asize sheet. This is defined in the .cdsplotinit file and, at least in the US, is likely to be 8.5x11 inches. **Orientation** chooses how the image is plated on the paper. You can see the image position on the right to see how things will be placed on the page.

Middle Section: The controls in the Plot Size section control how the plot is fit to the page. You may want to explore the **Center Plot** and **Fit to Page** options.

Bottom Section: Here you can select the number of copies to print and when to queue the plot (if you want to delay the printing until later). YOu can also select **Send Plot Only To File** to plot to a file instead of sending the file to a printer. The file name that you specify will appear in the directory in which Cadence was started. If you want email when the file has printed, select the **Mail Log To** option and give your email address. I seldom want extra email confirmation that the file has been printed, but you may like it.

Once you've selected your **Plot Options**, you can see them updated in the lower section of the **Submit Plot** dialog. In the example in Figure 3.15 you cansee that although I'm selecting the **CADE Printer**, I've also selected to **Plot To File** with a filename of **adder.ps**.

In the **Submit Plot** dialog you can select which schematic to plot. This should already be filled in for the schematic you were viewing when you selcted **Plot Submit** from the Composer menu. Note that you can select to plot the current **cellview** which is the entire schematic, or the **Viewing Area** which will plot the currently zoomed view in Composer. This is useful to zoom into sections of a schematic for printing. One final useful button is to unselect the **Plot With header** option so that the output is just a single page with the schematic and doesn't include an extra header sheet. If you're printing in an environment where header sheets are handy to keep hard copies organized, then leave the **header** option selected.

3.4.1 Modifying Postscript Plot Files

Although the Cadence Composer plotting interface produces good postscript files, there are some situations where I've found it useful to modify them. In particular, the postscript files produced by Composer are in color. This is fine if you're printing them on a color printer, or are including them in an on-line document that will be seen on a screen. However, if you're printing them on a black and white printer, the colors translate to grey values on the printer and the result is often a light grey schematic that is very hard to see. Also, the line size that is visible on the screen is sometimes not as bold on the printed page as you would like.

Unfortunately, I've found no easy way to modify the way that the postscript is produced by Composer. Instead, I've resorted to hand-editing the postscript files. Luckily postscript files are in plain ascii text and can be edited using any text editor.

To turn a color postscript output from Cadence into a black and white file (i.e. no grey scale), look for the following lines in your postscript file:

```
/slc {
  /lineBlue  exch def
  /lineGreen exch def
  /lineRed   exch def
  /color 1 def
  lineBlue 1000 eq {lineGreen 1000 eq {lineRed 1000 eq {
    /lineBlue  0 def
    /lineGreen 0 def
```

```
      /lineRed   0 def
    } if} if} if
  } def

  /sfc {
    /fillBlue  exch def
    /fillGreen exch def
    /fillRed   exch def
    /color 1 def
    fillBlue 1000 eq {fillGreen 1000 eq {fillRed 1000 eq {
      /fillBlue  0 def
      /fillGreen 0 def
      /fillRed   0 def
    } if} if} if
  } def
```

Change all the lines that read `/color 1 def` to be `/color 0 def` and the schematic will print in black and white with no color or grey scale approximation of color.

To change the thickness of the lines in the postscript file, look for the postscript function:

```
/ssls  { [] 0 setdash
          1 setlinewidth
} def
```

This sets the default line width to 1 point. If you increase the value of the line width to 3 or 4 you will get a bolder line in your schematic. These hacks can make much better looking schematics in hardcopy or to include in another document. Of course, depending on which other document you are including the graphics into you may need to convert the postscript into some other format such as pdf (ps2pdf) or jpg (I use Illustrator on a PC for this trick. The xv program on linux also works.).

3.5 Variable, Pin, and Cell Naming Restrictions

Although it's not obvious at the moment, all the simulation in the Cadence dfII environment is through Verilog simulators. So, although Composer allows wire and pin names that aren't legal Verilog names, your life will be *much* easier if you only use Verilog-compatable names within Composer. This means that names must start with a letter, and can then consist of letters, numbers, and underscores (the _ character) only. Do not use dashes or periods in names. Also, you should avoid all Verilog reserved words. The reserved words that are most likely to bite you are “input” and “output.” The complete set of reserved words is:

`and`	`for`	`output`	`strong1`
`always`	`force`	`parameter`	`supply0`
`assign`	`forever`	`pmos`	`supply1`
`begin`	`fork`	`posedge`	`table`
`buf`	`function`	`primitive`	`task`
`bufif0`	`highz0`	`pulldown`	`tran`
`bufif1`	`highz1`	`pullup`	`tranif0`
`case`	`if`	`pull0`	`tranif1`
`casex`	`ifnone`	`pull1`	`time`
`casez`	`initial`	`rcmos`	`tri`
`cmos`	`inout`	`real`	`triand`
`deassign`	`input`	`realtime`	`trior`
`default`	`integer`	`reg`	`trireg`
`defparam`	`join`	`release`	`tri0`
`disable`	`large`	`repeat`	`tri1`
`edge`	`macromodule`	`rnmos`	`vectored`
`else`	`medium`	`rpmos`	`wait`
`end`	`module`	`rtran`	`wand`
`endcase`	`nand`	`rtranif0`	`weak0`
`endfunction`	`negedge`	`rtranif1`	`weak1`
`endprimitive`	`nor`	`scalared`	`while`
`endmodule`	`not`	`small`	`wire`
`endspecify`	`notif0`	`specify`	`wor`
`endtable`	`notif1`	`specparam`	`xnor`
`endtask`	`nmos`	`strength`	`xor`
`event`	`or`	`strong0`	