CS/EE 5710/6710

Cadence V5 to V6 conversion notes
Digitial VLSI Chip Design CAD manual

Chapter 1: There are no significant V6 issues with Chapter 1 that I know about. The only minor one is that Verilog-XL is not used as a mainline Verilog simulator any more. NC_Verilog is the recommended Verilog simulator in V6.

Chapter 2: There are only very minor V6 issues with Chapter 2. The Command Interpreter Window (CIW) and Library Manager windows look ever so slightly different. The new windows look like the following:

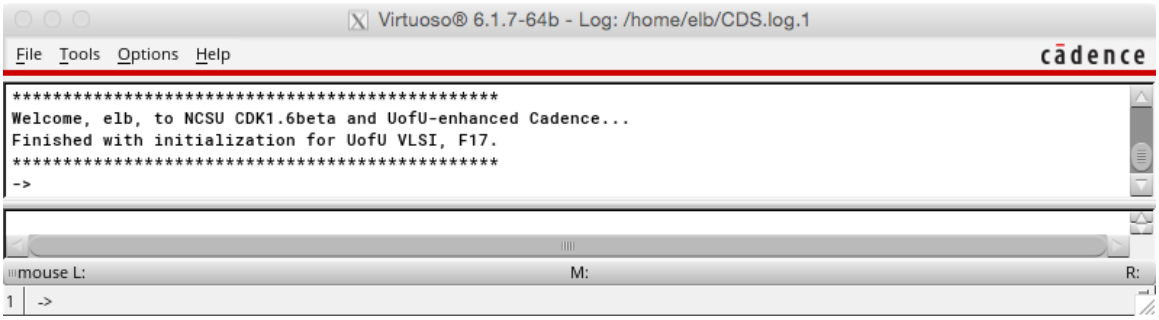


Figure a - Figure 2.1 from the text

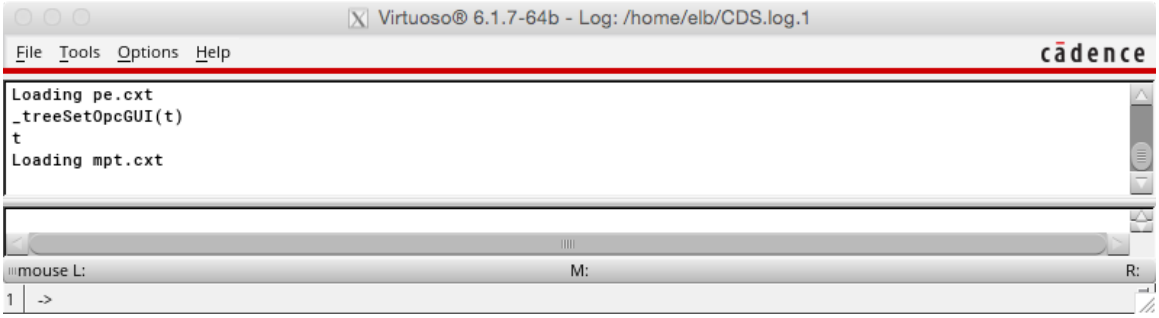


Figure b - You may see some extra commands in your CIW... They shouldn't harm anything.

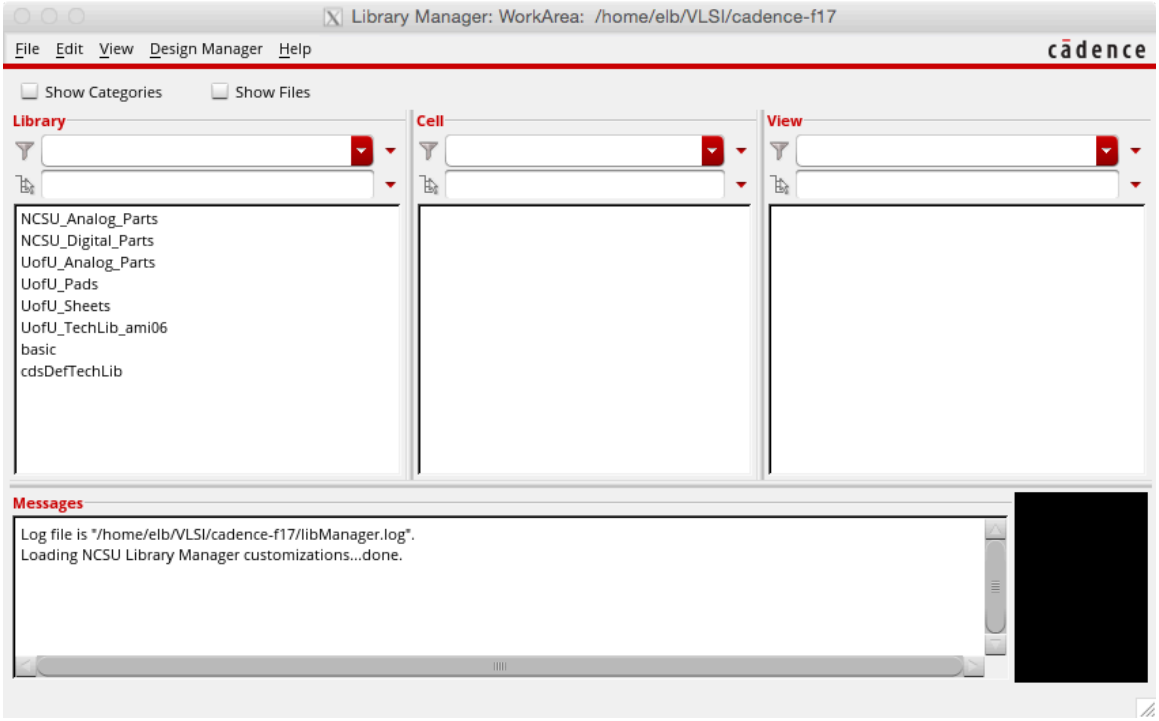


Figure b - Figure 2.2 from the text

Chapter 3: The Composer tool works very much the same as described in the text, but the interface in V6 looks a little different. Also, the integration with Verilog-XL has been removed. NC_Verilog is the preferred Verilog simulator in V6.

Here are some V6 versions of the figures from the book.

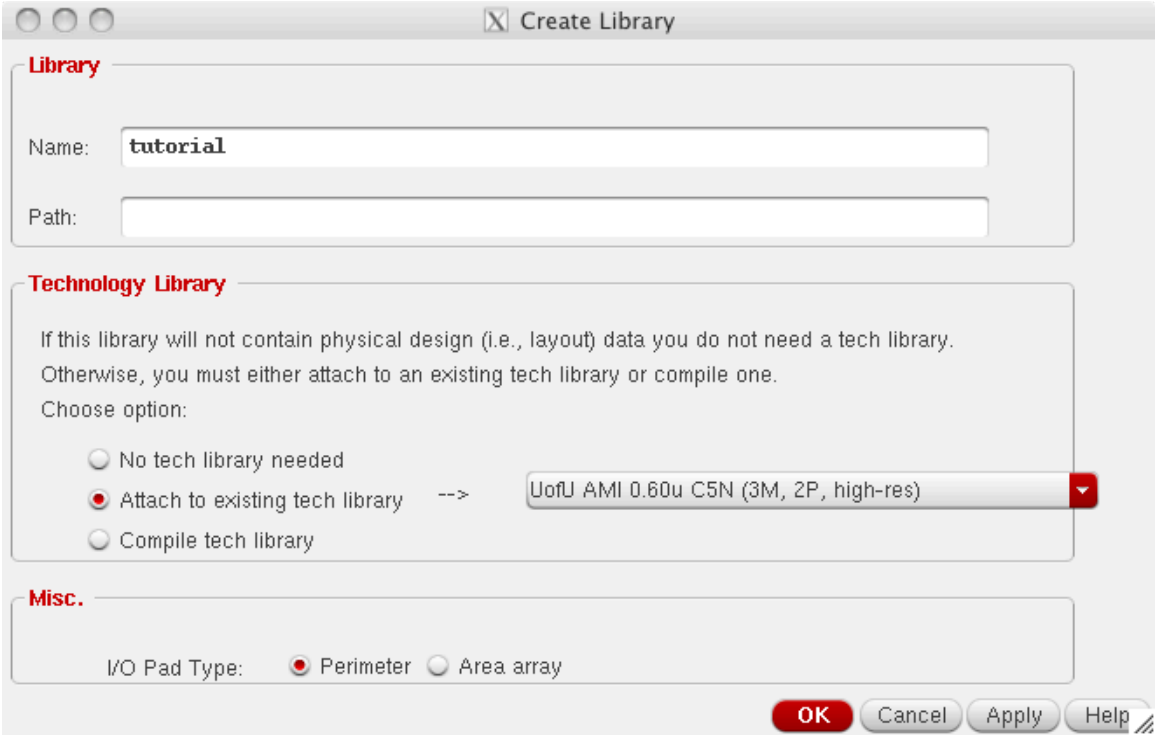


Figure c - Figure 3.1 in the text

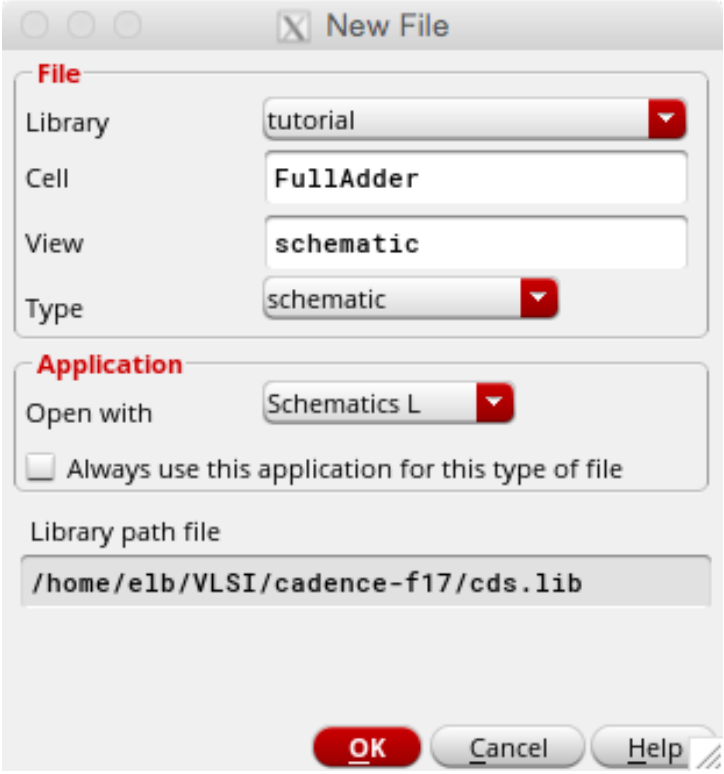


Figure d - Figure 3.2 in the text

In section 3.2.1: In the text is says **Add → Instance**. In the V6 menus this is **Create → Instance.** In fact, all the menu choices listed as in the **Add** menu in the text are in the **Create** menu in V6 (**wire**, **pin**, and **note** specifically). For the sheet titles, use **Edit → SheetTitle.**

In section 3.2.2 the symbol can be created with **Create → Cellview → FromCellview**. You can exit Composer with **File → CloseAll**.

In section 3.4, you print your schematic with **File → Print...**

The following are V6 versions of the figures from Chapter 3. Note that most of them are almost identical to the versions in the text, but I'll include them here just in case there are subtle differences. The main difference is that the main Composer window has a component browser on the left side, and an enhanced set of function widgets along the top.

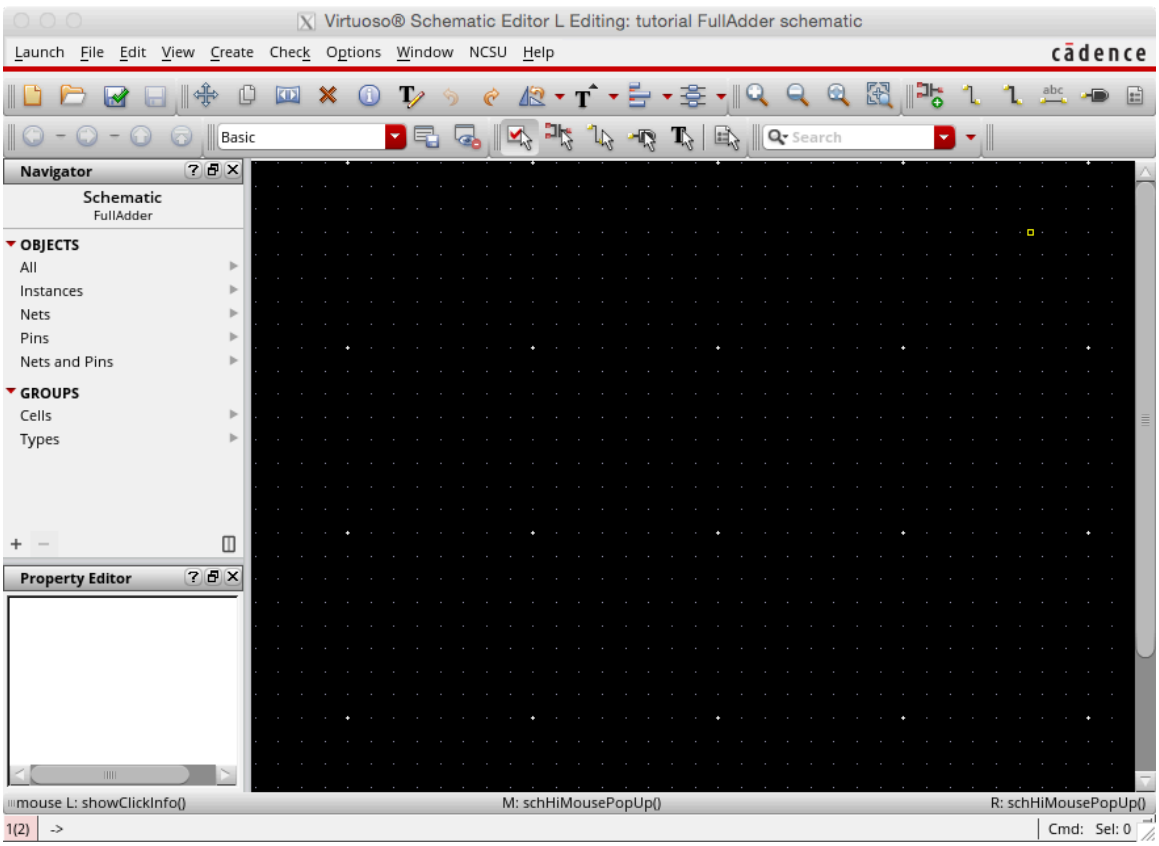


Figure e - Figure 3.3 from the text

I think the main dialog boxes that you see when using Composer look very much the same as in V5, but I'll include some screen captures here just in case.

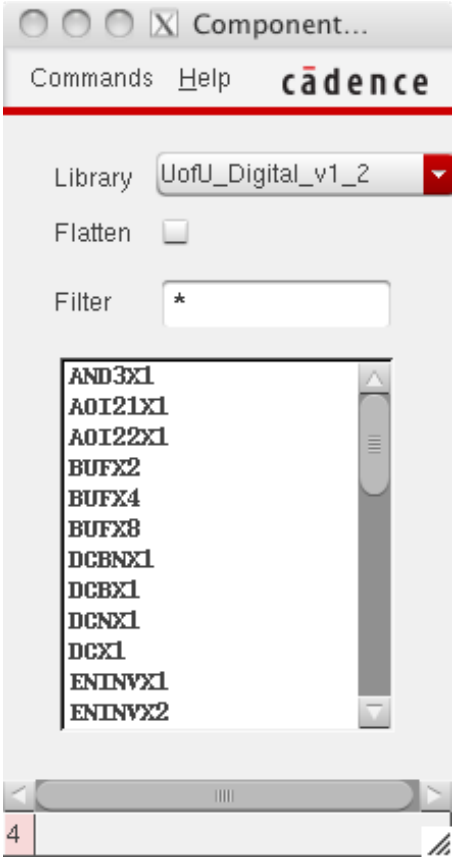


Figure f - Figure 3.4 from the text

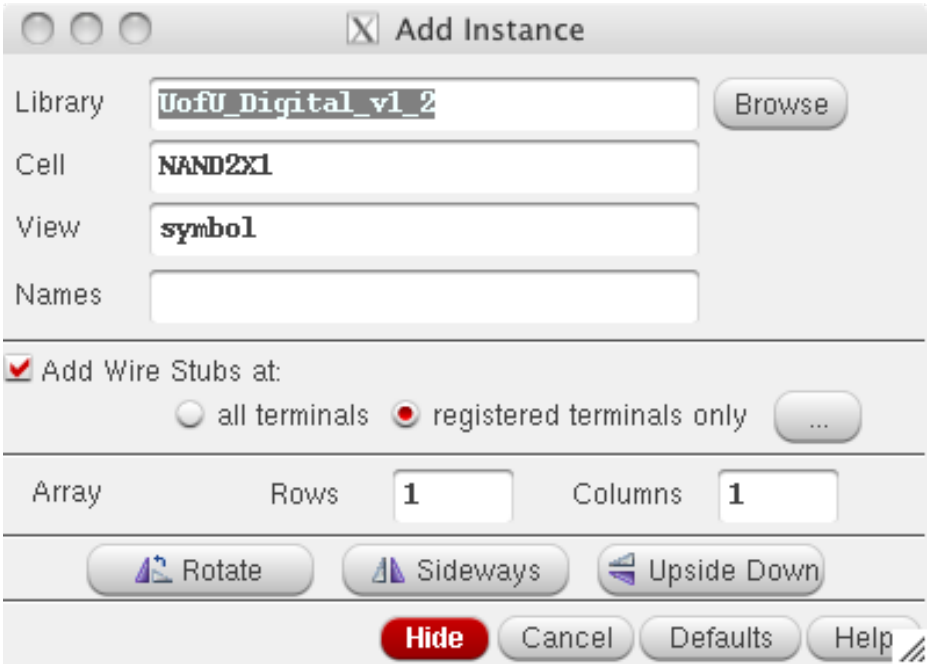


Figure g - Figure 3.5 from the text

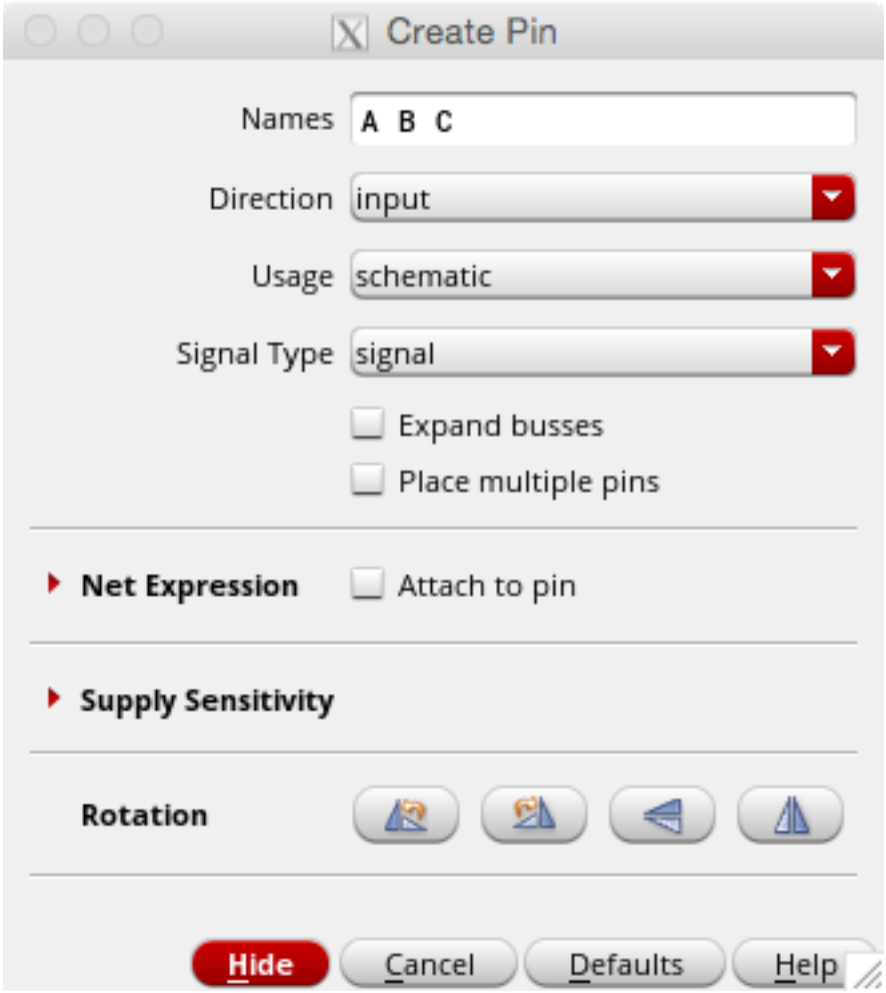


Figure h - Figure 3.6 from the text

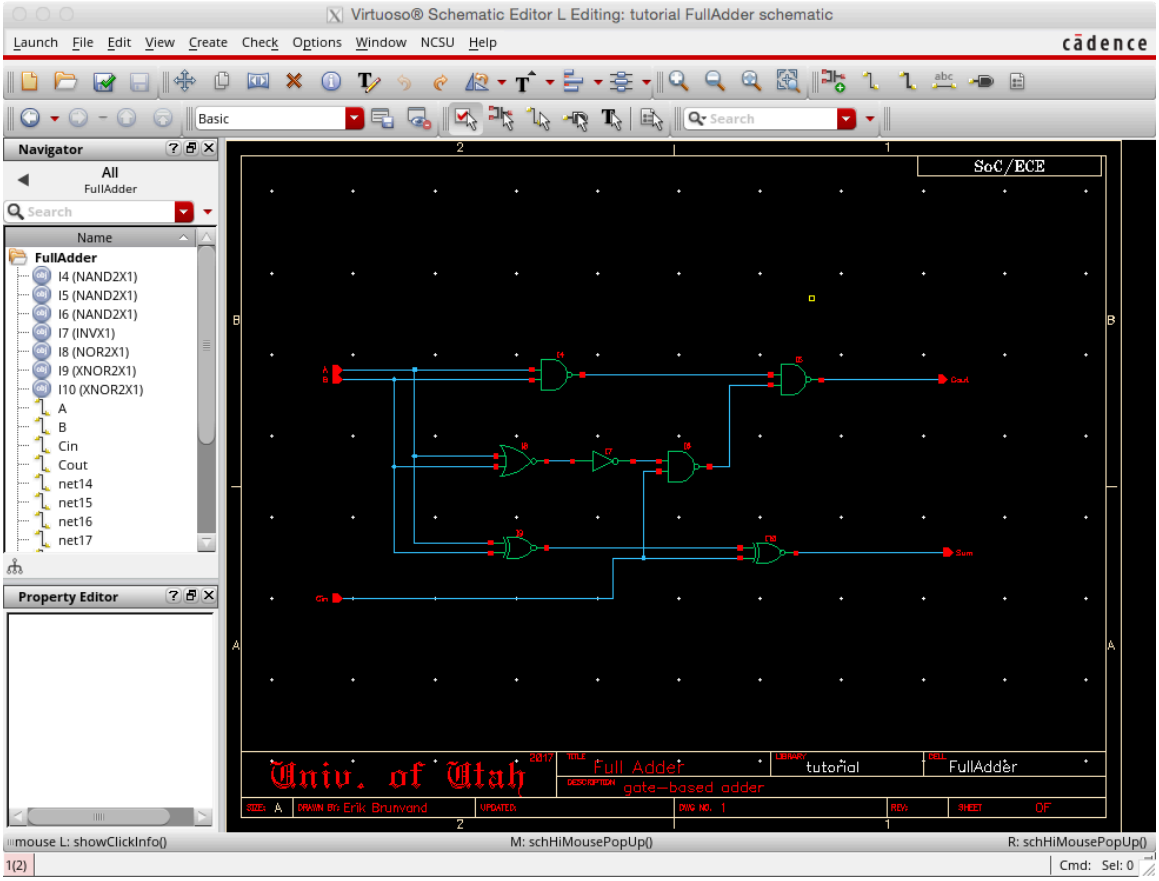


Figure i - Figure 3.7 from the text. You can see the new instance browser on the left side of the window. This lets you select components by name in the schematic.

Figures 3.8 and 3.9 in the text are pretty much the same in V6...

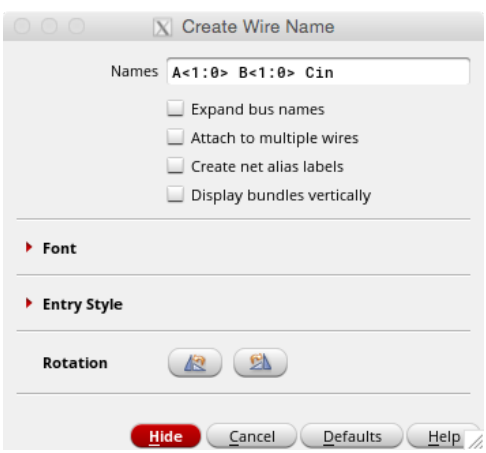


Figure j - Figure 3.10 from the text

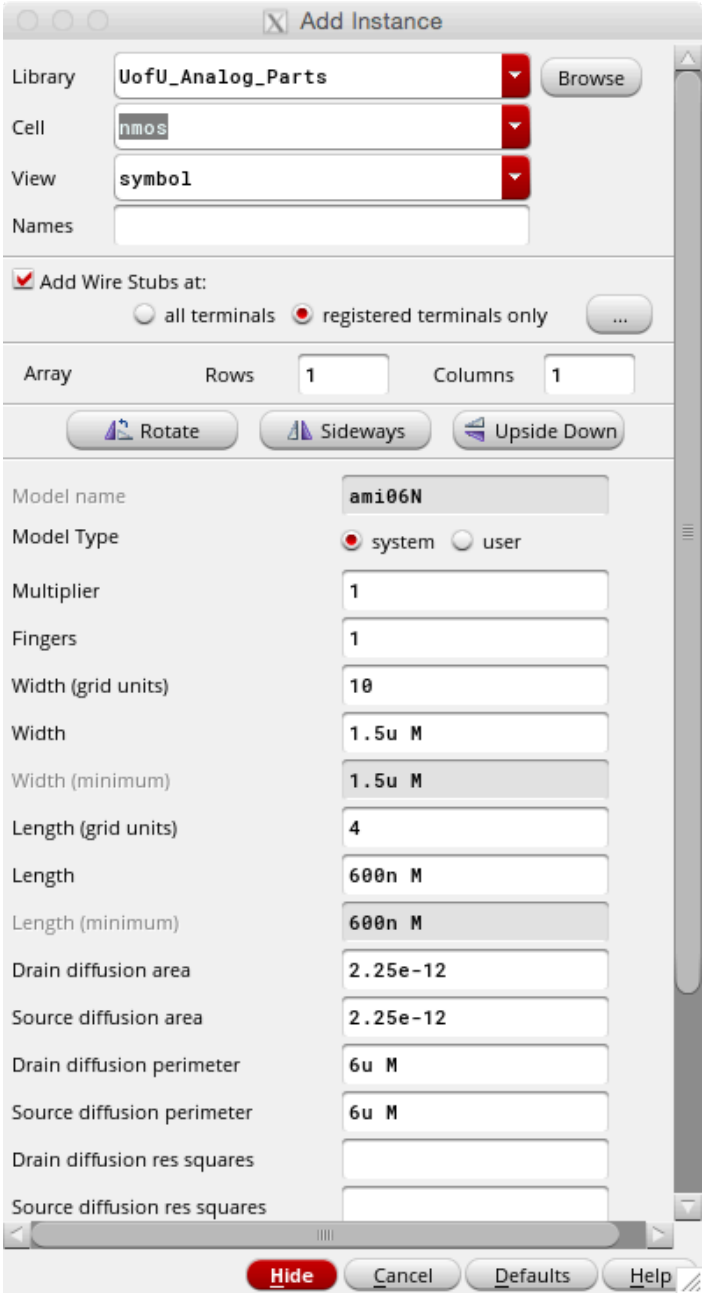


Figure k - figure 3.13 from the text

Figures 3.15 and 3.16 for plotting look very similar to the V5 figures in the text.

Chapter 4: The biggest change in Chapter 4 is that the Verilog-XL simulator is not recommended any more, so the automatic integration with Composer is now with NC_Verilog. Verilog-XL was very specifically a Verilog-1995 simulator. It would not interpert any Verilog code that was not included in the 1995 standard. NC_Verilog includes all the new Verilog-2001 features, and is the mainline simulator that will continue to be upgraded to new versions of Verilog. So, you can safely skip all the stuff in Chapter 4 related to Verilog-XL.

In section 4.1.2 there are only mild differences between the V5 and V6 interfaces to NC_Verilog. The biggest difference is that to get to the NC_Verilog dialog box you need to go through Launch -> Plugins -> Simulation -> NC-Verilog. This gets you to the dialog box in Figure 4.22 of the text.

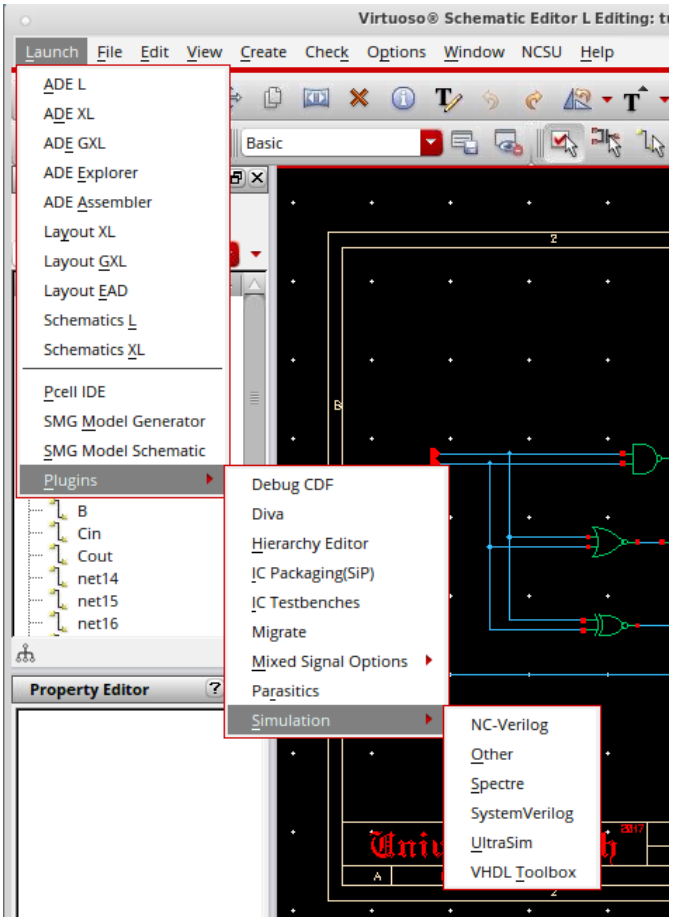


Figure l - Menus to get to the NC_Verilog dialog box

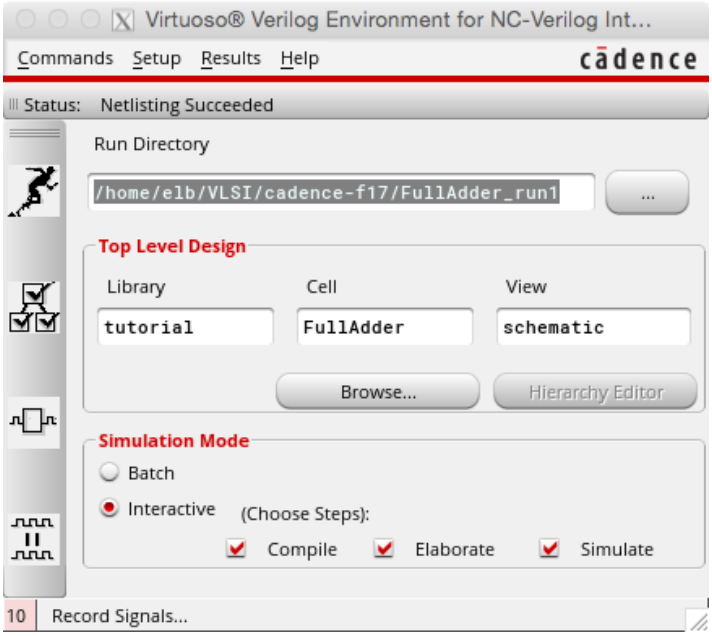


Figure m - Dialog box for initializing simulation with NC_Verilog (Figure 4.22 in your text)

Here are some of the figures that are different enough for you to notice anything.

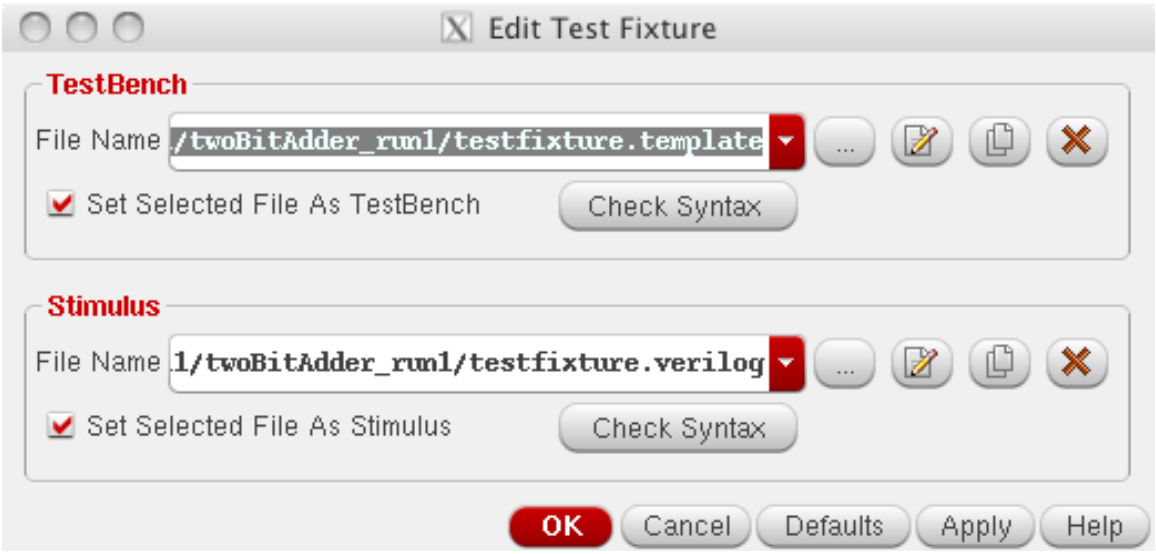


Figure n - Figure 4.24 in the text. Notice that you are also allowed to change the TestBench file in this version of the dialog. In practice you should not have to change either of the files names in this dialog box.

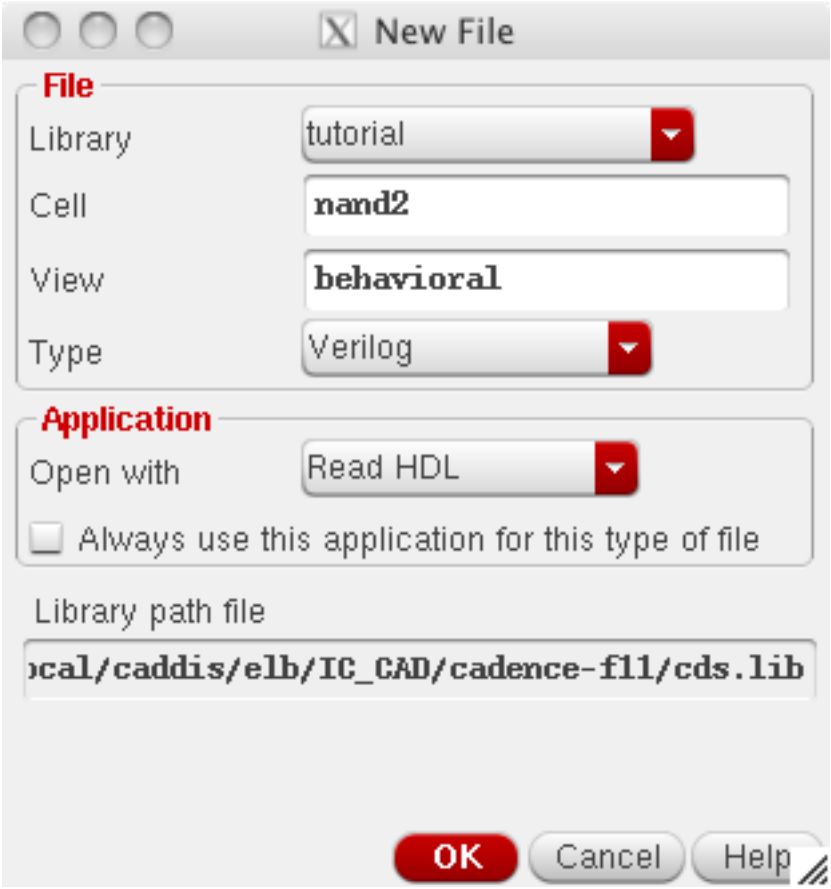


Figure o - Figure 4.28 in the text. The only difference here is that you can give the new file a "type". You probably won't have to worry too much about this label. You should make sure that the type matches what you expect this cell view to be. In practice, the tool is pretty good at getting the default correct.

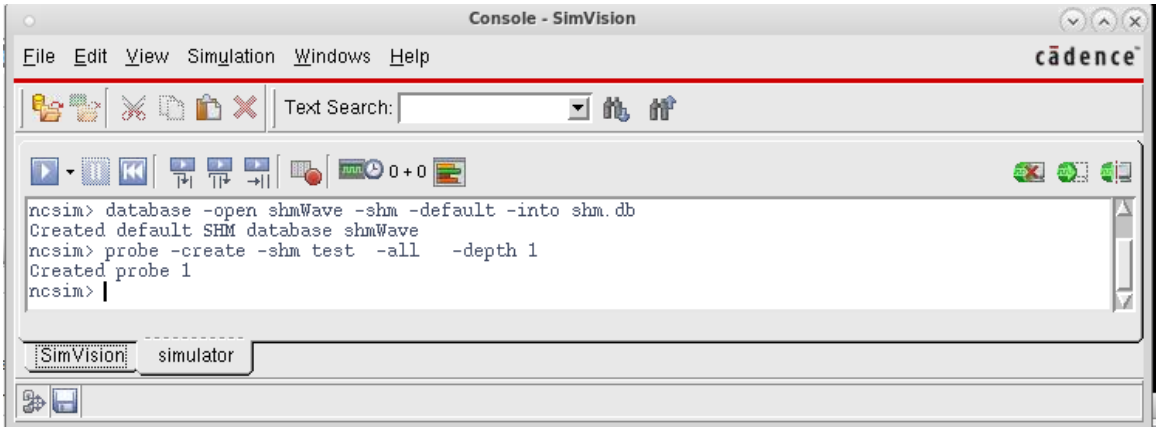


Figure 12 - SimVision console – you can start up simulations from here, but you probably want to do that from the waveform viewer instead. If your simulation is self-checking and you don't need to see waveforms, you can fire up the simulation here and check your outputs in the simout.tmp file.

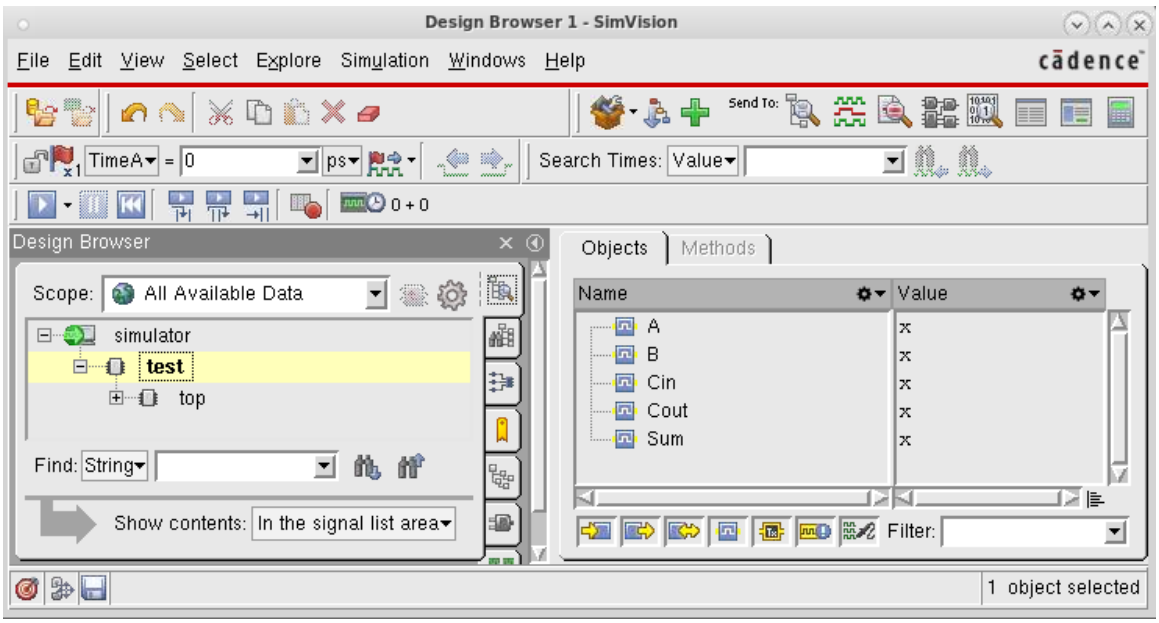


Figure 13 - SimVision Design Browser. You can select signals and send them to the waveform viewere from here.

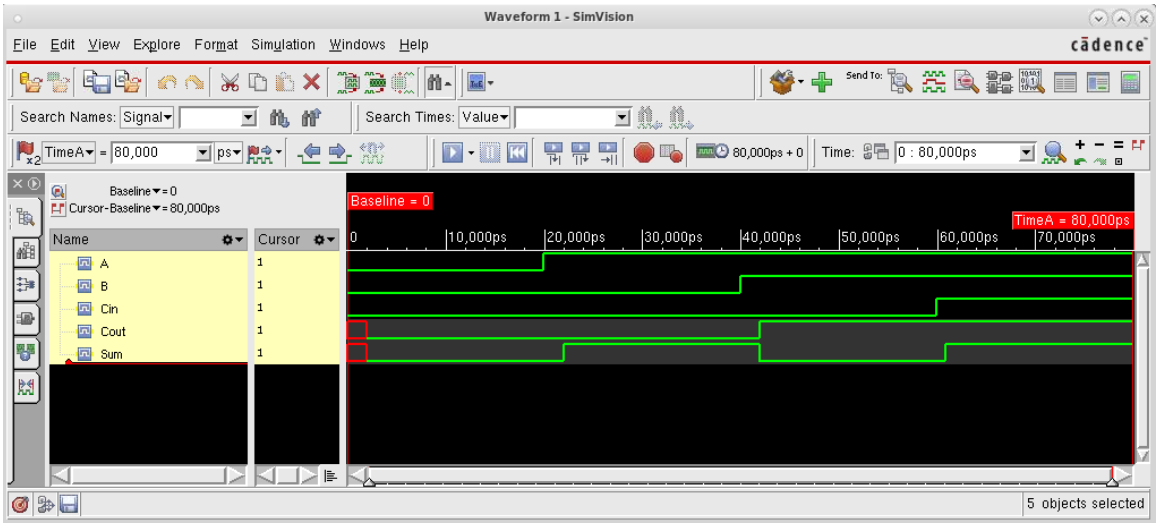


Figure 14 - SimVision Waveform Browser.

The other figures in Chapter 4 should be similar enough to not cause any problems.

Chapter 5: The main Virtuoso window will look a bit different in V6. The main window is shown here. The main differences are that it now includes the Layer Selection Window as a pane on the left part of the Virtuoso window rather than as a separate window. Also, there are extra command widgets both on the top and bottom of the window. You can hover your mouse over those to see what they do.

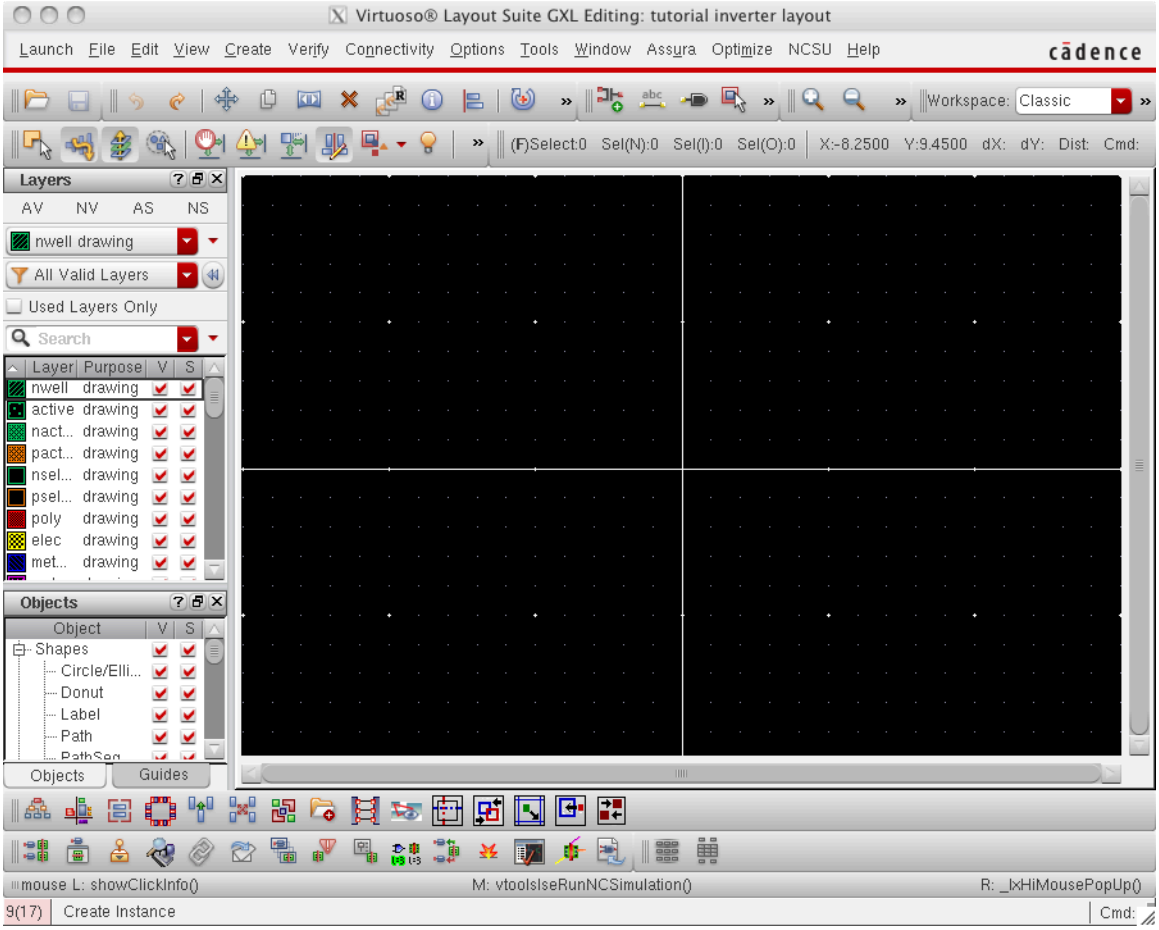


Figure p - This doesn't correspond with any particular figure in the text. It should probably come before Figure 5.4 in the text.

When you get to page 116 where the text talks about creating a contact, you need to substitute "**Create Via**" instead in V6. The Create Via dialog box is a little fancier than what is shown in Figure 5.6 in the text. Mostly you can ignore the extra fancieness and just select the via that you want, and the rows and columns that you wns if you're making a multiple-cut via.

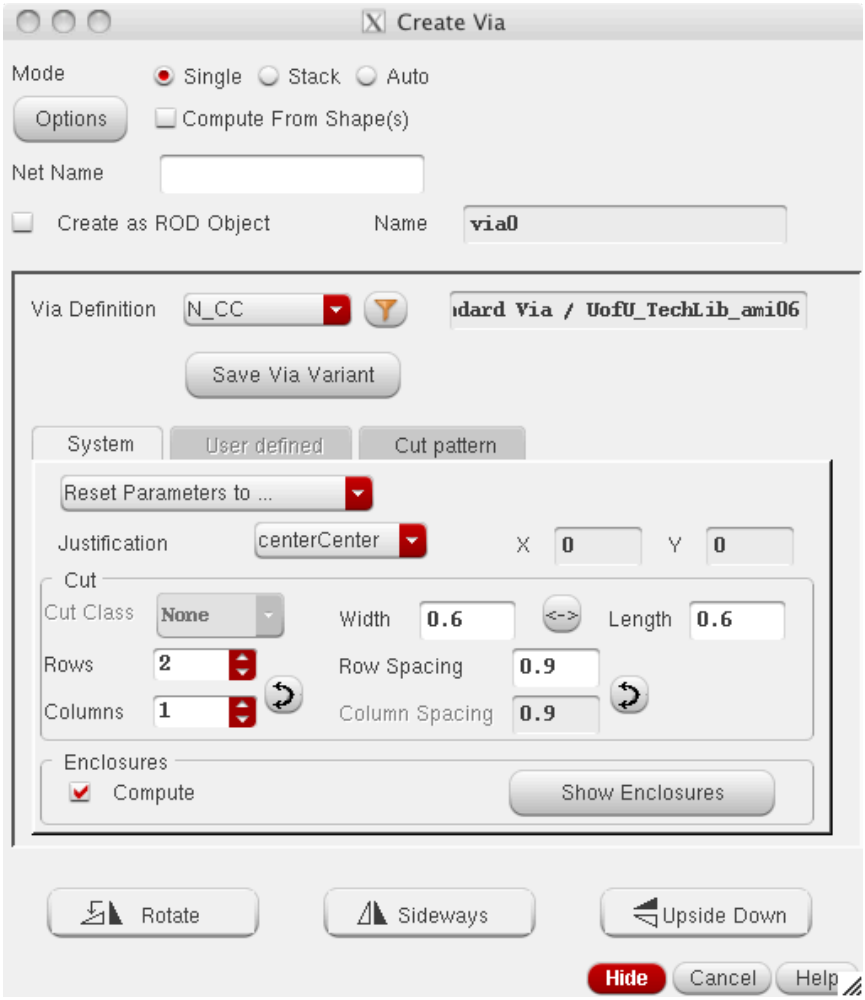


Figure o - Figure 5.6 in the text.

In the **Create → Path** dialog (Figure 5.12) I haven't played with paths enough to know how to set the "**Change to Layer**" option, or whether that still works. In the Shape Pin dialog box (Figure 5.16), the main change is that addition of the **Connectivity** button that has to do with the router that you might use later. You should probably leave the connectivity to the default of **Strong**.

In section 5.4.1 on design rule checking, the dialog for finding all the DRC errors has a few more choices of what type of errors to be interested in. This can help you narrow down what you're looking for more quickly.

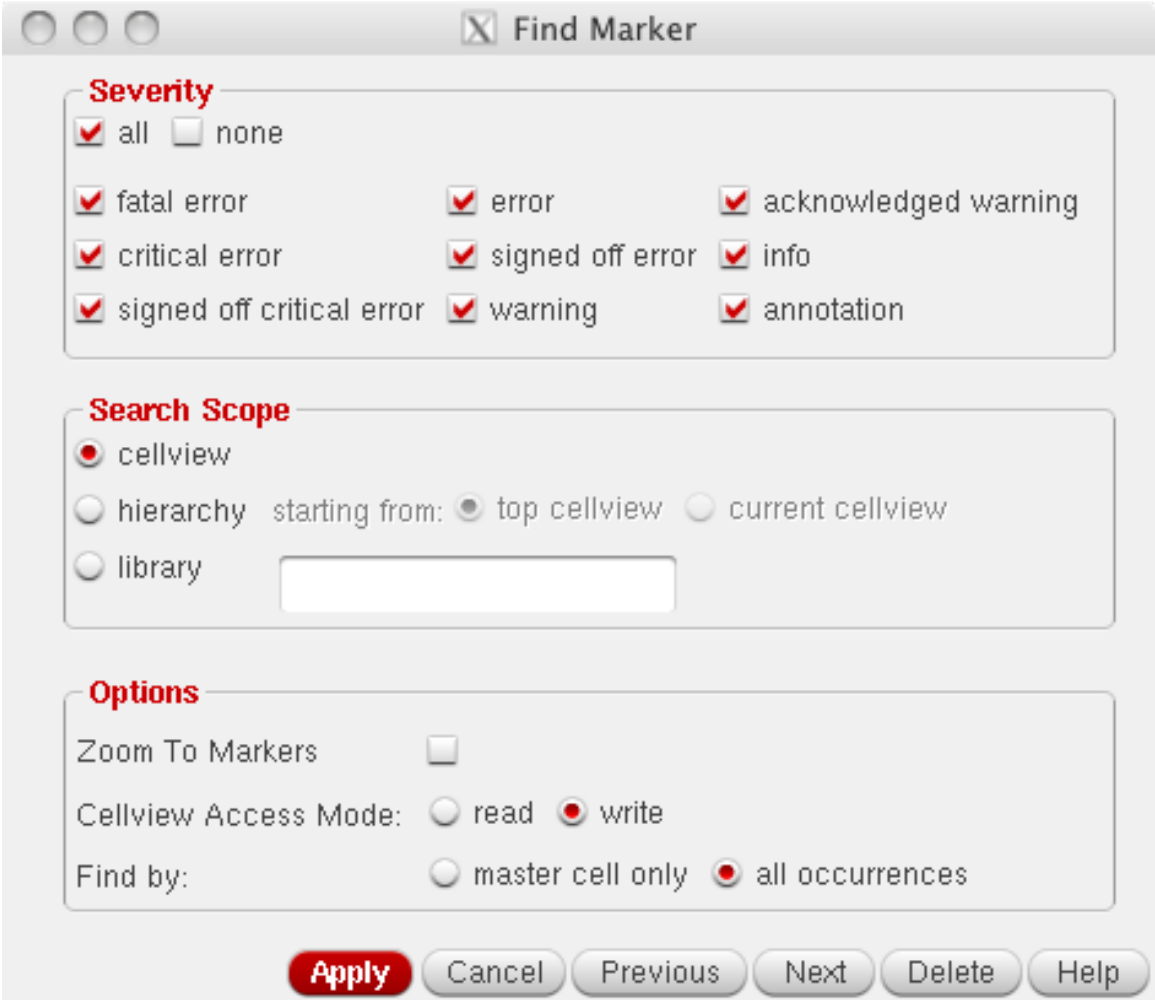


Figure p - Figure 5.27 in the text

In section 5.5 on the Extraction/LVS process, the biggest change is that when the LVS process reports completion, it also reports the result of the LVS. This is a very nice change!

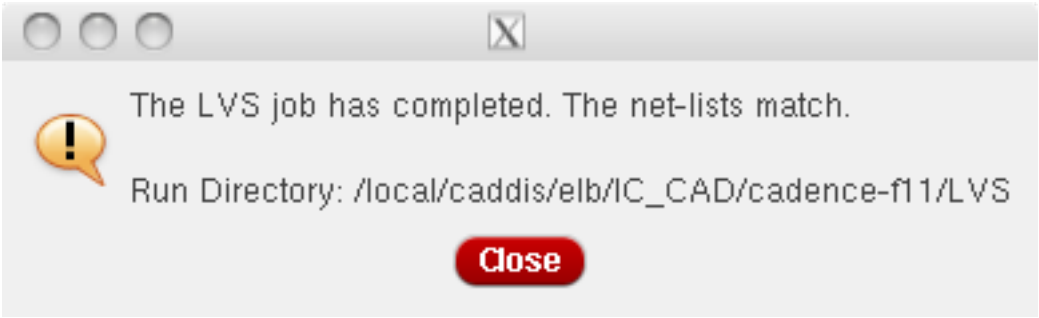


Figure q - Figure 5.35 from the text

Chapter 6: There should be very few V6 issues with Chapter 6. The overall template measurements for the UofU standard cell template have not changed.

Chapter 7: The Spectre simulator itself has not changed much in V6. There are two new interfaces for the Analog Design Environment (ADE) though. The one we used in V5 is now called ADE-L and is the one you should use if you're following along with the text. ADE-XL and ADE-GXL are not covered in the CAD text. The interaction with ADE-L is largely the same as described in the text. The waveform window has a few enhancements like better notation of what signals the waveforms represent on the left of the window, and a preview-view of the waveform in the small window at the top.

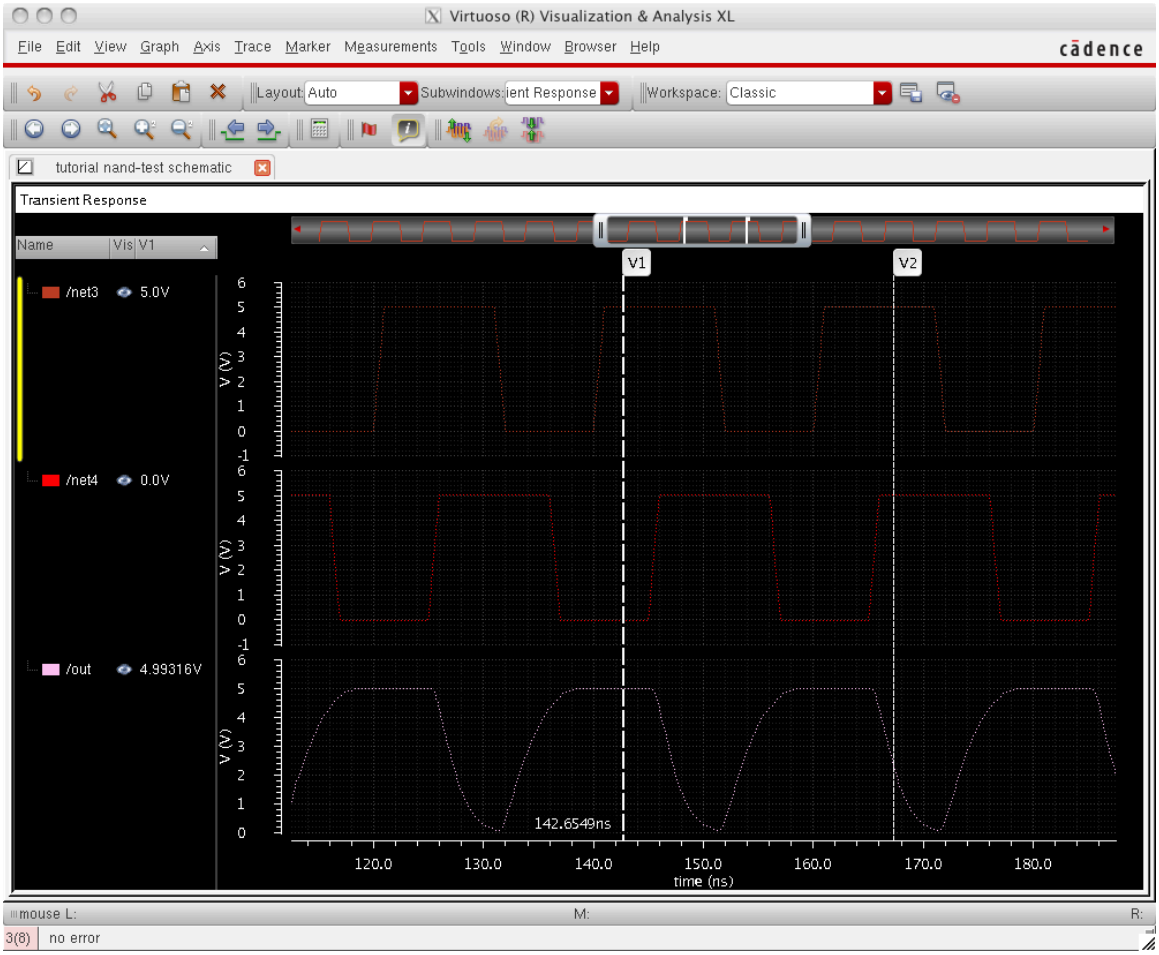


Figure r - Figure 7.11 in the text

One thing that's not quite as described in the text is the setting of markers. The dialog box for the **Marker → CreateMarker** menu is shown here.

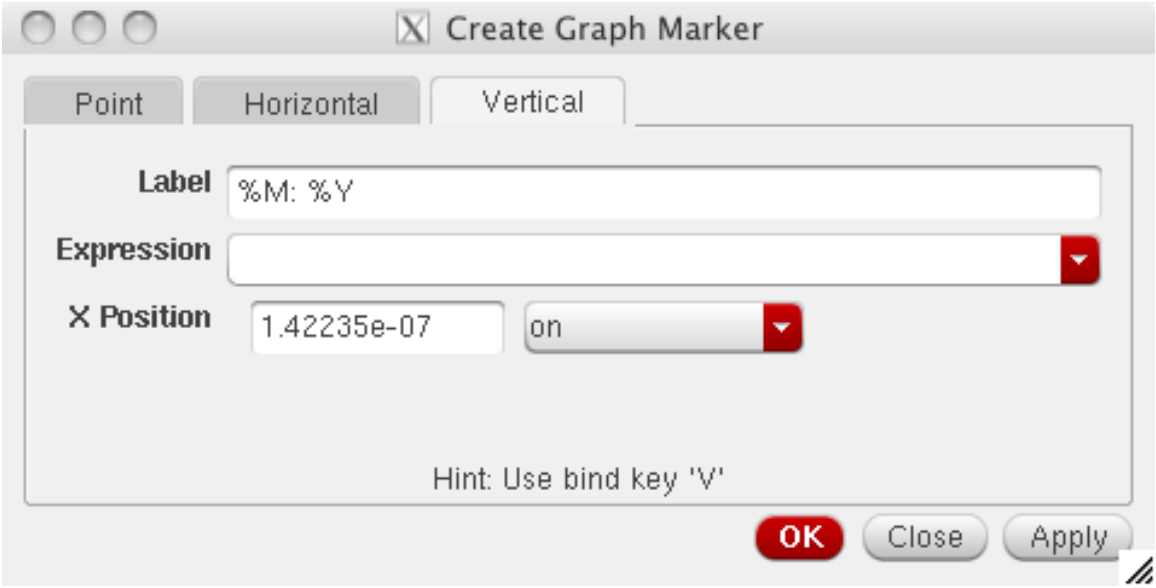


Figure s - This is the dialog box for creating a marker line (vertical or horizontal) in the waveform viewer. It doesn't correspond to any figure in the text, but would be near figure 7.11.

When creating a config view as described in section 7.3, the Hierarchy Viewer has tabs for the tree and table views which is a nice feature.

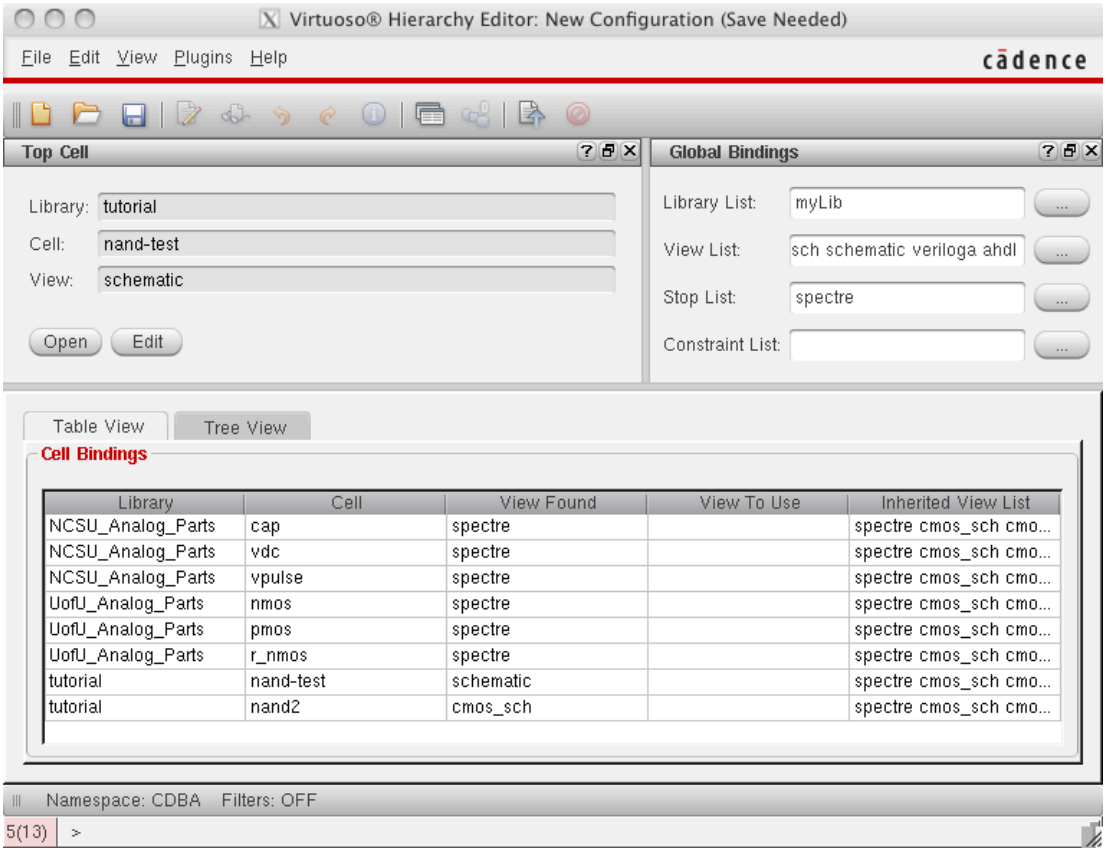


Figure t - Figure 714 in the text.

In section 7.5.1 on parametric simulation, the dialog to set the parameters has all the same information as before, but it's organized slightly differently in the dialog box.

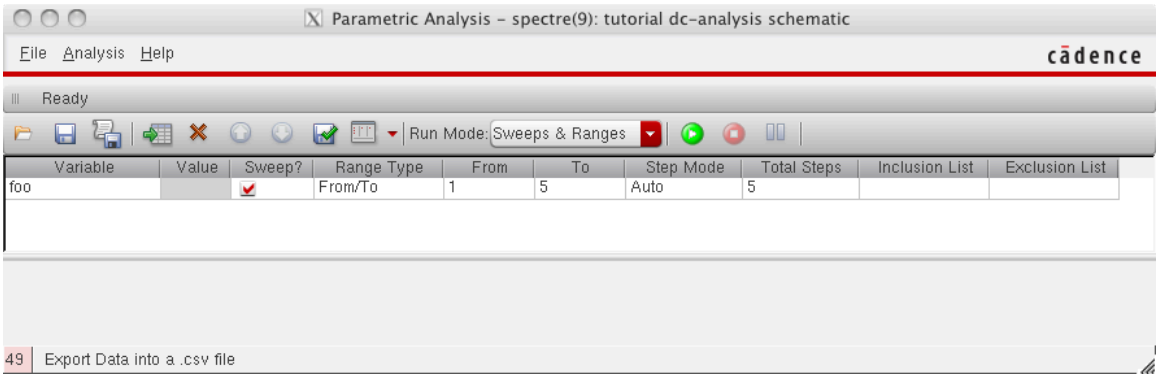


Figure u – Figure 7.34 in the text.

In section 7.6 on power measurement, the ADE-L calculator is described as a way to do the power calculation. The calculator looks very different in V6 than it did in V5. The interface is shown here.

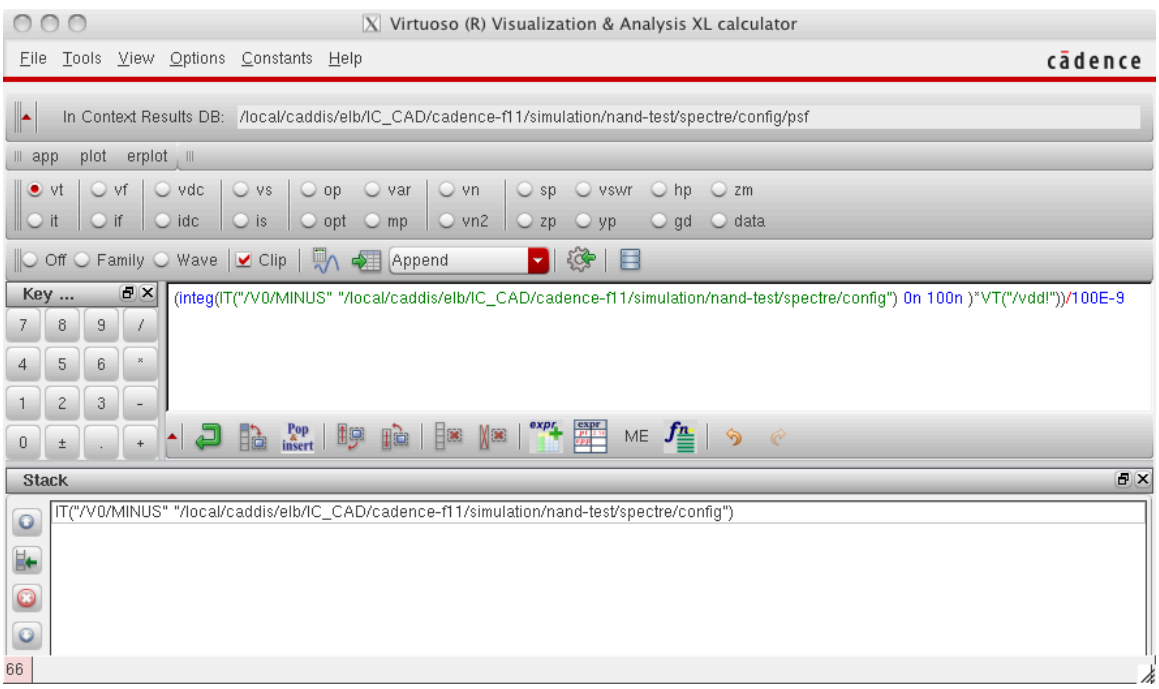


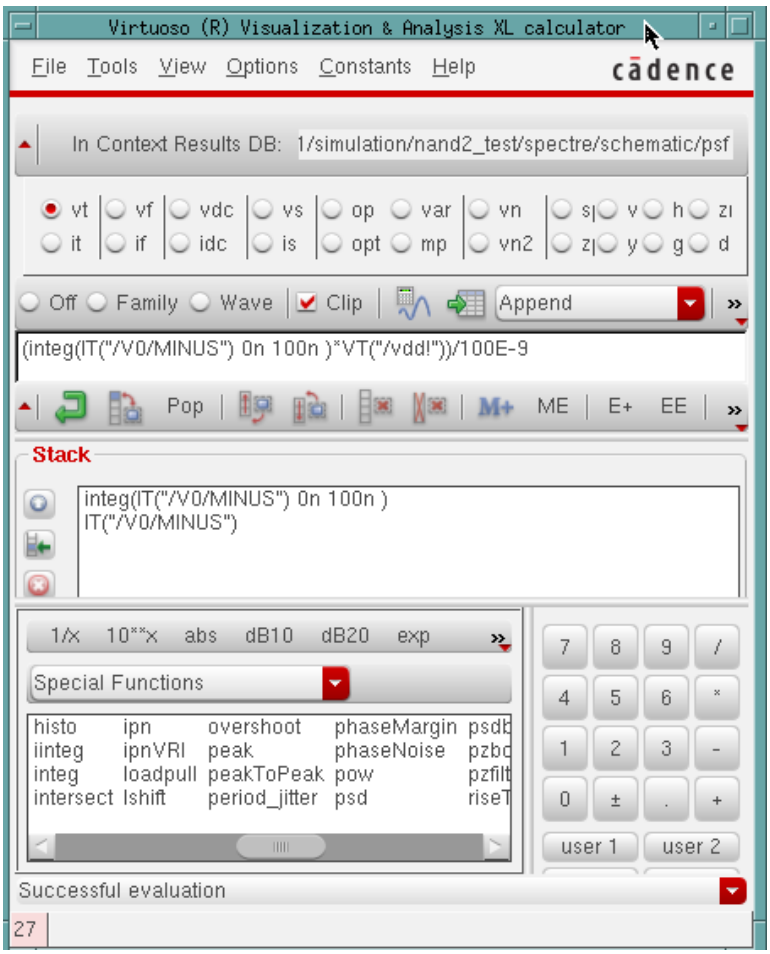
Figure v - Figure 7.39 in the text.

It's basically the same thing, but not every button is labeled with the same text as shown in the book.

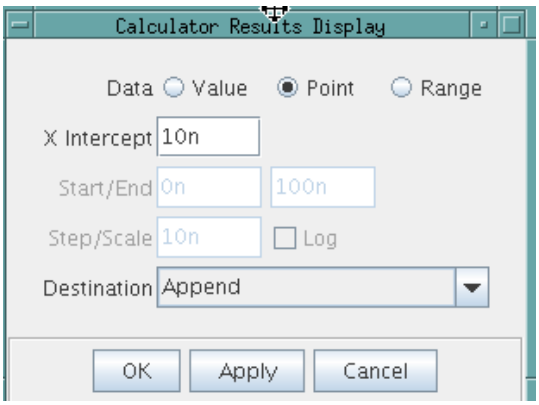
The biggest difference is that there is no "eval" or "print" button in the new interface. Instead there is a "plot" button that will plot the results in your waveform window, and a "table" button that looks like a green arrow pointing at a grid that will output the value into a table. Also, the scientific notation in the calculator is a little different. Instead of

100EEX-9 you would enter 100E-9 or even 100n. Also note that the green u-shaped arrow is the “put the current value on the stack” button (the default mode is still RPN).

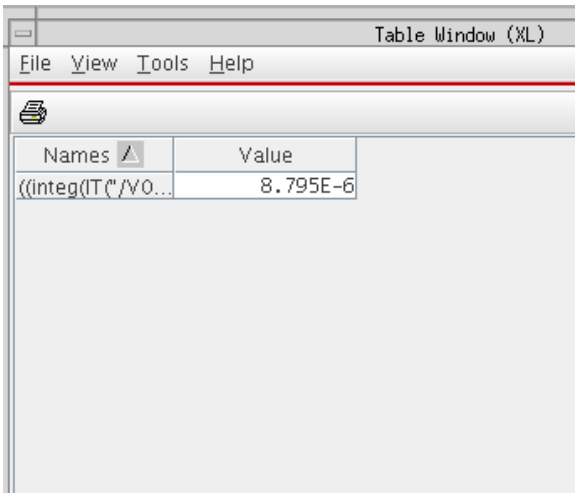
This is what the calculator looks like when set up to do a similar calculation as described in section 7.8 of the CAD book. Note that I resized the calculator so the individual panes shifted around a little from the previous figure.



The button just to the right of the “Clip” check box is the “plot” button that will plot the value in your waveform window. The next button to the right is the “print to table” button that you can use to print the scalar value of your expression. When you press that button you get the following box where you can select to print just one point, or a whole range of points that represent the expression. In this case, the expression evaluates to a scalar, so you get only one value no matter how many times you print it.



The result that is printed looks like the following:



In this case the value is 8.795 uWatts.

Chapter 8:

The Digital VLSI Chip Design book describes in detail the process of characterizing a cell library. That is, take the extracted layout for a set of standard cells, and run analog simulations on those cells to characterize the electrical/timing behavior in way that can be used by HDL synthesis. This means understanding the propagation delays, the internal power, the input capacitance, and many other aspects of the behavior of the cells. The result is a "liberty format" .lib file that describes all this electrical and timing behavior. The sections in Chapter 8 that deal with Liberty format (.lib format) have not changed. That format is still the same

Section 8.2 describes the use of the Encounter Library Characterizer (ELC) tool. Unfortunately, that tool is no longer supported by Cadence. The new library characterizer tool from Cadence is called Liberate. This tutorial replaces section 8.2 for use with Liberate. Because Liberate is a completely different tool than ELC, this section of the guide will be a little longer and more detailed than others in this guide.

Section 8.2.1 describes how to generate the input netlist for the characterization. This process is the same for the new tool. Up to the creation of the netlist using the ADE-L tool (page 233) you can follow this exact process from the book.

The differences start with the modifications to that netlist file to make it compatible with Liberate. The main difference is that Liberate seems to strongly prefer vss as the name of the ground node in the circuit. Also, Liberate allows the use the "global" parameter in the Spectre input file to define certain nodes (typically the power and ground nodes) as global in scope. With the vss and vdd nodes described as global, the individual "subckt" descriptions no longer need to include the vdd and vss arguments. An example of a Liberate-modified netlist for two cells (INVX1 and NAND2X1) replacing Figure 8.17 is:

```
simulator lang=spectre
global vss vdd

subckt INVX1 A Y
M1 (Y A vdd vdd) ami06P w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
    pd=15.0u m=1 region=sat
M0 (Y A vss vss) ami06N w=3u l=600n as=4.5e-12 ad=4.5e-12 ps=9u \
    pd=9u m=1 region=sat
ends INVX1

subckt NAND2X1 A B Y
M2 (Y A vdd vdd) ami06P w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
    pd=15.0u m=1 region=sat
M3 (Y B vdd vdd) ami06P w=6u l=600n as=9e-12 ad=9e-12 ps=15.0u \
    pd=15.0u m=1 region=sat
M0 (Y B net12 vss) ami06N w=6u l=600n as=9e-12 ad=9e-12 \
    ps=15.0u pd=15.0u m=1 region=sat
M1 (net12 A vss vss) ami06N w=6u l=600n as=9e-12 ad=9e-12 \
    ps=15.0u pd=15.0u m=1 region=sat
ends NAND2X1
```

There is a script in the F17 bin directory called scs2liberate that should automatically make the modifications to the netlist generated by ADE-L to make it compatible with Liberate. The use f this script is:

```
scs2liberate <input-file> <output-file>
```

You should check the output of this tool carefully to make sure it looks right. It has been tested, but not on a huge number of input files! To make things easier later in the process, you should name the output file **`libcells.scs`**.

Now everything is different from the book! The new tool does the same thing as ELC, but it is a completely different tool with a different interface.

In your VLSI directory (or wherever you want run Liberate from), make a recursive copy of the **`Liberate`** directory from **`/uusoc/facility/cad_common/local/class/6710/F17/cadence`**. This will copy a number of directories and setup files. The following Linux commands do the trick (make sure you include the "dot" at the end of the cp command).

```
cd ~/VLSI
cp -r /uusoc/facility/cad_common/local/class/6710/F17/cadence/Liberate .
```

In that new **Liberate** directory you will see the following:

1. A **lib** directory – that's where the generated **<library>.lib** file will be generated.
2. A **netlist** directory – put your **libcells.scs** file in this directory.
3. A **templates** directory – edit the **UofU_Cell_Defs.tcl** file in this directory to include cell descriptions for each of the cells you want to characterize. The **UofU_Templates.tcl** file has definitions for the timing and power templates that will be used for characterization. You probably don't need to modify these unless you are using a different technology than ON Semi C5N.
4. A **UofU_Cells.tcl** file – edit this file to make a list of the cells that you want to characterize in this run. This could be a list of every cell described in **templates/UofU_Cell_Defs.tcl**, or it could be a subset if you just want to try a few.
5. A **userdata** directory – edit the **userdata.lib** file to reflect the areas and footprints of each of the cells in your library.
6. A **tcl** directory – Edit the **UofU_Char.tcl** file in this directory to change the name of the library that the tool generates. If you don't modify this, the tool will generate a file named **Lib6710_XX.lib** by default. The **settings.tcl** file in this directory has configuration commands for the Spectre simulator. You won't need to modify this file at all.
7. A **models** directory that has the Spectre model files for the ami06N and ami06P transistors used by in your netlist.
8. A **run.sh** shell script that calls the cad-lib Liberate script with the appropriate input files, and makes a copy of the log information in a **Liberate.log** file.

Let's take a closer look at each of these steps.

Characterization Netlist: Start with your **libcells.scs** file that describes the transistor-level netlist for the cells you'd like to characterize (the equivalent of Figure 8.17 in the book). Put this file into the **Liberate/netlist** directory.

Cell Descriptions: Edit **templates/UofU_Cell_Defs.tcl** to add definitions for each cell that you will be characterizing. You can continue to add cells to this file as you add cells to your library. This is the central place where the cell interfaces are described. The actual cells being characterized on any particular run of the tool are only those listed in the **UofU_Cells.tcl** file. As an example, here are the descriptions for INVX1 and NAND2X1. This is essentially the same as the interfaces extracted by ELC and shown in Figure 8.19 in the book, but you'll have to type them in by hand.

```
if {[ALAPI_active_cell "INVX1"]} {
     define_cell \
              -input { A } \
             -output { Y } \
             -pinlist { A Y } \
            -delay delay_template_5x5_X1 \
            -power power_template_5x5_X1 \
           INVX1
}

if {[ALAPI_active_cell "NAND2X1"]} {
     define_cell \
              -input { A B } \
              -output { Y } \
              -pinlist { A B Y } \
             -delay delay_template_5x5_X1 \
             -power power_template_5x5_X1 \
             NAND2X1
}
```

The "if" statement says to ignore the cell description if it's not currently active in the tool. This is for efficiency of execution. After the "if" each cell is described by a **`define_cell`** command where the inputs, the outputs, the argument list (in the order used in the netlist), the delay and power templates to use, and the name of the cell are described. The delay and power templates are defined in the **`UofU_Templates.tcl`** file. You can look in that file to see what they are and how they're defined. Basically there are versions of the delay_template and power_template that are used for X1, X2, X4, and X8 (and larger) cells. The difference is that the cells with larger drive strengths are characterized over a larger range of capacitive loads.

Sequential cells are further characterized for constraints such as setup and hold times. The template for constraints is called **`constraint_template_5x5`**. An example of a flip flop definition is:

```
if {[ALAPI_active_cell "DCBX1"]} {
      define_cell \
              -clock { CLK } \
              -async { CLR } \
              -input { D } \
              -output { Q QB } \
              -pinlist { CLK CLR D Q QB } \
             -delay delay_template_5x5_X1 \
             -power power_template_5x5_X1 \
             -constraint constraint_template_5x5 \
             DCBX1
}
```

You need to add a description for each new cell that you add to your library.

Now you need to add the cells that you want to be characterized on this run of the tool to the **`UofU_Cells.tcl`** file. This is just a list of cell names. The list could be all the cells described in the **`templates/UofU_Cell_Defs.tcl`** file, or just a subset of them (if

you want to try out a new cell without characterizing the whole library again, for example). The format of this file is:

```
#
# List the cells that you want to include on
# this characterization run.
# The cell definitions should be in the
# templates/UofU_Cell_Defs.tcl file.
set cells {
       INVX1
       NAND2X1
}
```

User data: This file (**`userdata/userdata.lib`**) has information about the area and footprint of your cells. This is similar information to the **`footprints.def`** file described in the book on page 242, but in a different format. The area is straightforward – it's the area of the cell. I measure the area from the center of the lower left contact in the GND line to the center of the upper right contact in the VDD line. The footprint is used to group cells into similar functionality.

As described in the book, cells with the same footprint can actually have different areas. For example, all your INVXn cells should have the same footprint (INV), but they'll each have different areas. The footprint information lets the place & route tool choose suitable replacements as required for achieving speed and power goals. An example of the userdata.lib file is:

```
library (LIB6710_XX) {
     cell (INVX1) {
            area : 129.6;
           cell_footprint : "INV";
      }
      cell (NAND2X1) {
             area : 194.4;
            cell_footprint : "NAND2";
      }
}
```

Characterization commands: The commands that drive the tool are in the two files in the tcl directory: **`settings.tcl`** with deep dark secret settings for the tool (I got this file from Cadence and haven't dissected exactly what every line means), and **`UofU_char.tcl`** which drives the actual characterization run with your files. The only thing you probably have to change in this file is the name of the library you want to end up with. Even this isn't essential – you can always edit the default **`Lib6710_XX.lib`** file to be a different name. But, it only takes a second to modify the LIBNAME field in this file. You can also see the sequence of commands that will drive the actual characterization.

The only other thing you might want to change in this file is to select a different set of model files for the transistors. By default you'll get the "typical" parameters for the ON

Semi C5N process (called ami06N and ami06P because ON Semi used to be named AMI Semiconductor). This is almost always what you want.

But, you could also try simulations with best-case and worst-case conditions. You could change the PROCESS variable to "ff" for "fast-fast" to get the best-case performance, and "ss" for "slow-slow" to get the worst-case behavior as described by ON Semi. You could also go to mosis.com and get the extracted models for a recent run on the process and use that model if you like. That might be a more up to date version of the model parameters. An example of a mosis.com extracted model is also in the **`models/spectre`** directory (**`T89Y.scs`**).

Running Characterization: Now that you've modified all the relevant files you can actually run the Liberate tool. You could run it directly from the cad-lib script, but there's a **`run.sh`** script in your new **`Liberate`** directory that runs the tool, with the **`UofU_Char.tcl`** as the input file, and forking the console output to a **`Liberate.log`** file so you can look at the log and make sure you didn't have any serious errors or warnings in the process. You can run this shell script, but you'll probably have to say explicitly where it is using the dot notation (which says to look for it in your current directory):

```
./run.sh
```

If everything was specified correctly, this will result in a **`Lib6710_XX.lib`** (or whatever named you changed it to) in the lib directory of your Liberate run directory. This .lib file can be used directly by the place & route tools, but needs to be converted to binary format for the Synopsys HDL compiler Design Compiler.

As with the older ELC tool, this characterization will spawn a bunch of Spectre simulations for various conditions, and automatically check the timing and power of the cell. These data are collected into the proper Liberty format and eventually output into the .lib file. Because a large number of Spectre simulations are used for the characterization, this can take a while for a large library with a lot of complex cells.

Converting the .lib file: To convert the .lib into a binary .db file for Design Compiler, you'll use a tool called Library Compiler from Synopsys. Call this with the **`syn-lc`** script. The process is to read the .lib file, then output that library as a binary version in a .db file. This is as described in the CAD book in Section 8.4, but use the **`syn-lc`** library compiler script directly rather than going through the design compiler interface. This is the recommended flow in the latest Synopsys tools. Here's a transcript of the simple 2-cell version:

```
[elb@lab2-20 lib]$ syn-lc
Using setup-synopsys from S17
Assuming your OS is amd64
You are now set up to run the synopsys tools.

Working directory is /home/elb/VLSI/Liberate/lib

                              Library Compiler (TM)
                                  DesignWare (R)
                 Version M-2016.12-SP3 for linux64 - Apr 13, 2017
                       Copyright (c) 1988 - 2017 Synopsys, Inc.
    This software and the associated documentation are proprietary to Synopsys,  Inc.
This software may only be used in accordance with the terms and conditions
  of a written license agreement with Synopsys, Inc. All other use, reproduction,
             or distribution of this software is strictly prohibited.

Initializing...
lc_shell> read_lib Lib6710_XX.lib
Reading '/home/elb/VLSI/Liberate/lib/Lib6710_XX.lib' ...
Warning: Line 1, The 'default_inout_pin_cap' attribute is not specified. Using 1.00. (LBDB-172)
Warning: Line 1, The 'default_input_pin_cap' attribute is not specified. Using 1.00. (LBDB-172)
Warning: Line 1, The 'default_leakage_power_density' attribute is not specified. Using 0.00.
(LBDB-172)
Warning: Line 105, Cell 'INVX1', The cell_leakage_power attribute of the 'INVX1' cell is
redundant
        and not used in the leakage_power modeling. (LBDB-644)
Warning: Line 215, Cell 'INVX1', pin 'A', The pin 'A' does not have a internal_power group.
(LBDB-607)
Warning: Line 229, Cell 'NAND2X1', The cell_leakage_power attribute of the 'NAND2X1' cell is
redundant
        and not used in the leakage_power modeling. (LBDB-644) Technology library
'Lib6710_XX' read successfully
1
lc_shell> write_lib Lib6710_XX -o Lib6710_XX.db Wrote the
'Lib6710_XX' library to '/home/elb/VLSI/Liberate/lib/Lib6710_XX.db' successfully
1
lc_shell> exit Memory usage for this session 19 Mbytes.
CPU usage for this session 0 seconds ( 0.00 hours ).

Thank you...
[elb@lab2-20 lib]$
```

You'll see that there are a number of warnings – typically related to the cell leakage power calculation. You can safely ignore these warnings because the leakage power in our process is negligible anyway. The Liberate tool also seems to not generate every possible attribute that the library compiler understands. You can safely ignore this too.

You can now use your converted library with Synopsys HDL synthesis as described in the CAD book as described in Chapter 9.

Chapter 9: Synthesis with Synopsys design compiler, and especially using the syn-script-tcl template from the class directory should work the same way as described in the CAD book. The version of design compiler has changed, but the script should work the same way. I haven't explored the differences in the Design Vision graphical interface carefully, but everything appears to be very similar to what's in the CAD book.

The Cadence RTL Compiler seems to also have a similar interface to what's in the book. We're now using v10.10-s209_1, but the script works with the new version, and the gui looks very similar.

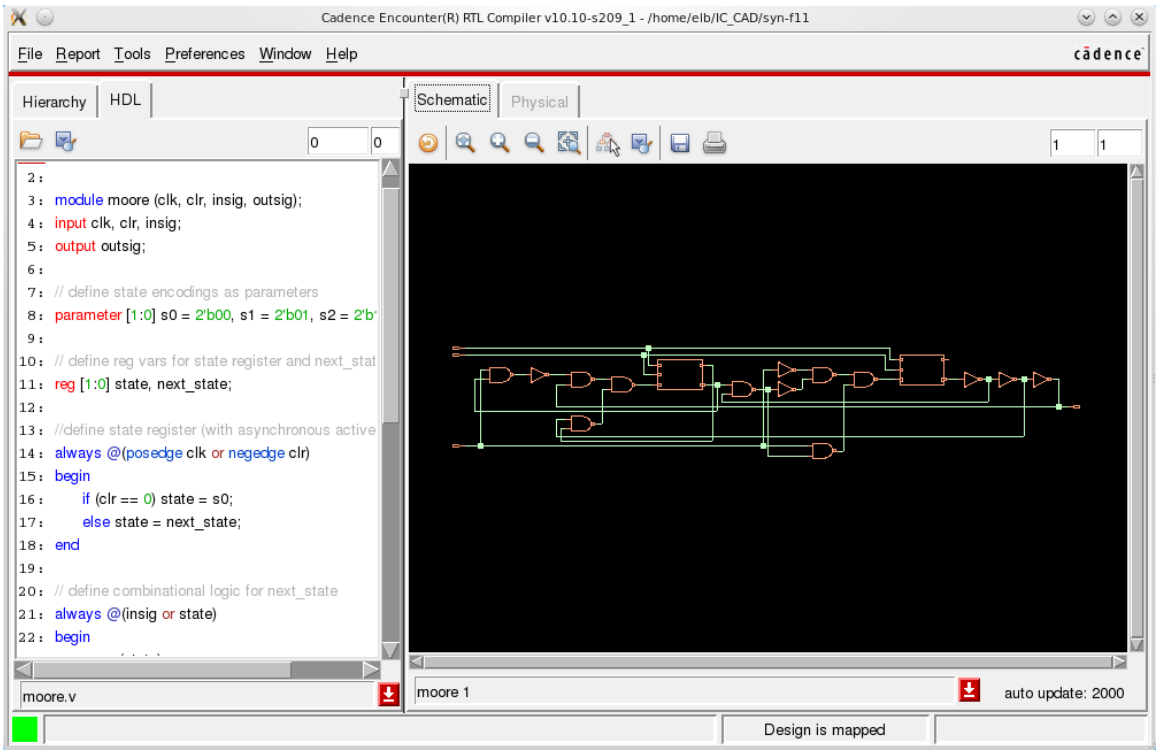


Figure w: Figure 9.27 in the text

The procedure for reading the structural Verilog back into the Cadence Composer schematic tool is similar to what's in the book, with minor differences in the dialog boxes.

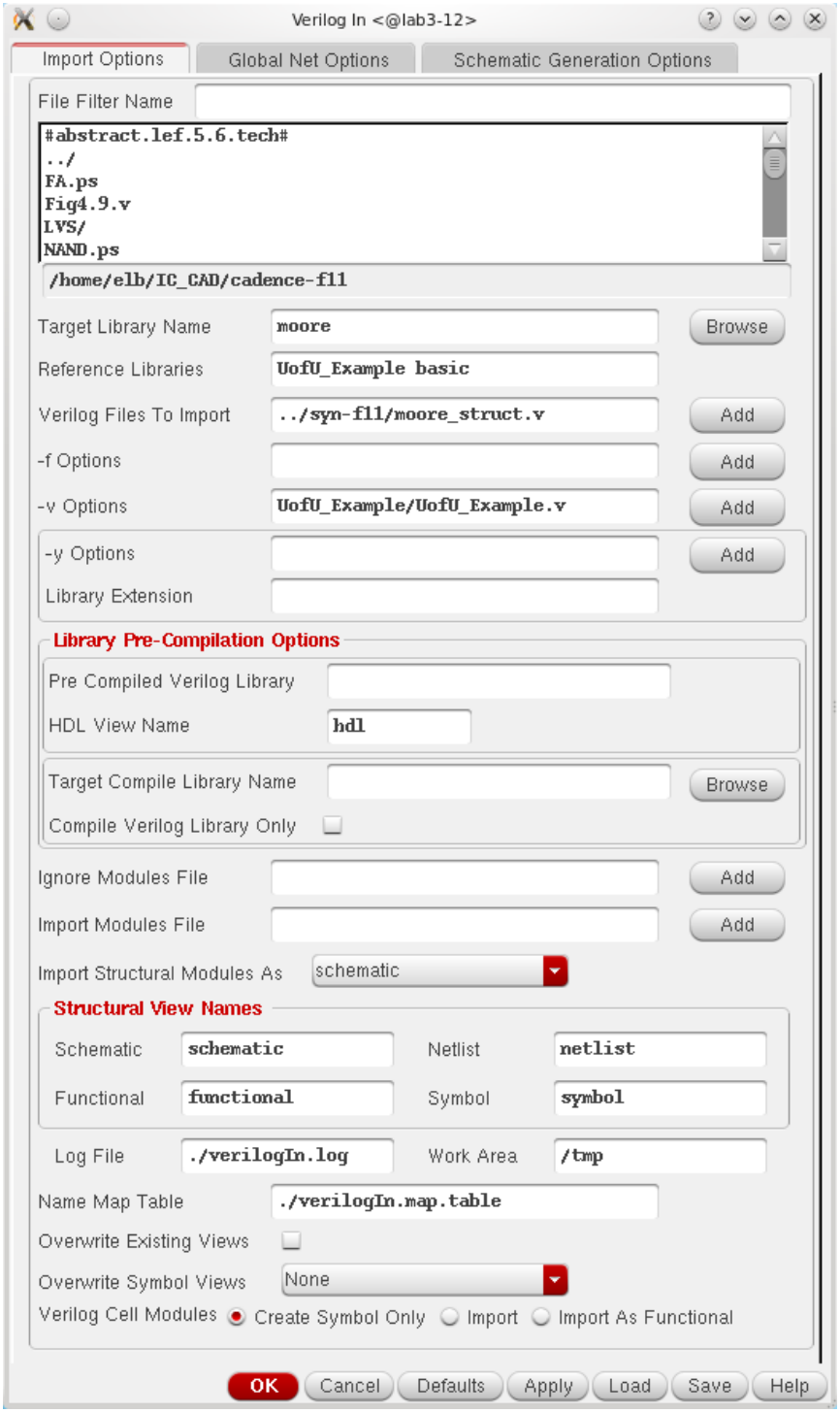


Figure x: Figure 9.29 in the text

It isn't shown in the CAD book, but when you read the structural Verilog back into the schematic tool, you should get the following dialog box if everything worked correctly. This shows that the cells that were referenced in the structural view were correctly found in the Reference Libraries.

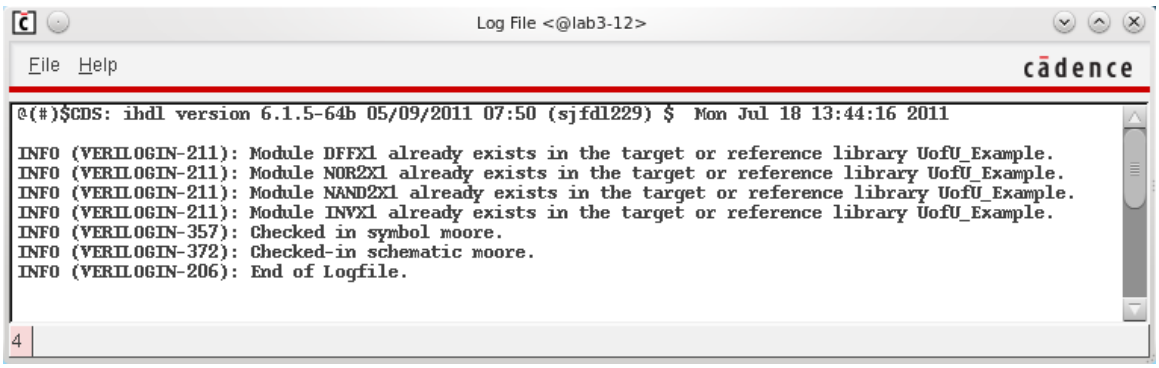


Figure y: Import Verilog log file showing that the library cells are found in the reference library (UofU_Example in this case)

The imported Verilog, now looking like a schematic, looks very similar to the book's version, but with the new Composer Schematic interface.

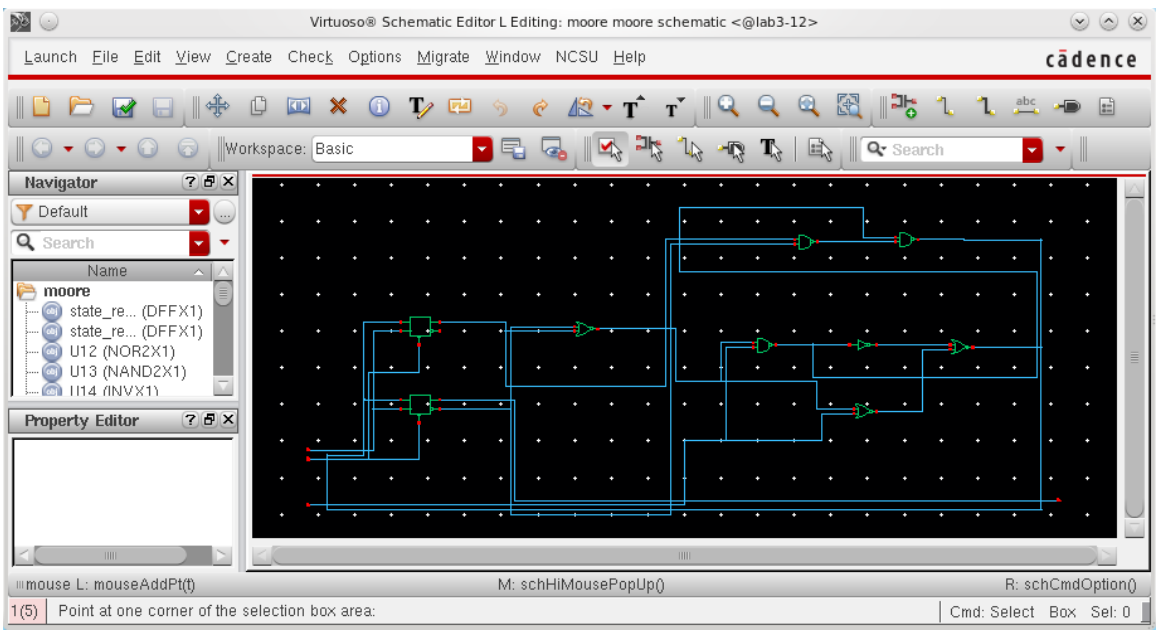


Figure z: Figure 9.30 in the text.

Chapter 10: The interaction with the Abstract tool is very similar to what's in the book. The new version of Abstract has very few changes to the interface.

One change is to the "abstract" step (Section 10.4). In that dialog box, in addition to making sure that the "site" is selected as "core" (because you're generating abstracts for core cells), you need to make sure that in the Blockages tab, not only are the routing and contact/via layers described as Detailed Blockages, you also need to make sure that the "Pin Cutout" box is checked for all layers. This combination of settings means that the tool will generate blockages for routing layers that you use inside your cells, but not make material that is part of a port also be part of a blockage. This lets the place & route tool

have more leeway in where it chooses its connection point on the port, and doesn't result in as much pure blockage in the cell, which makes the other wires easier to route.

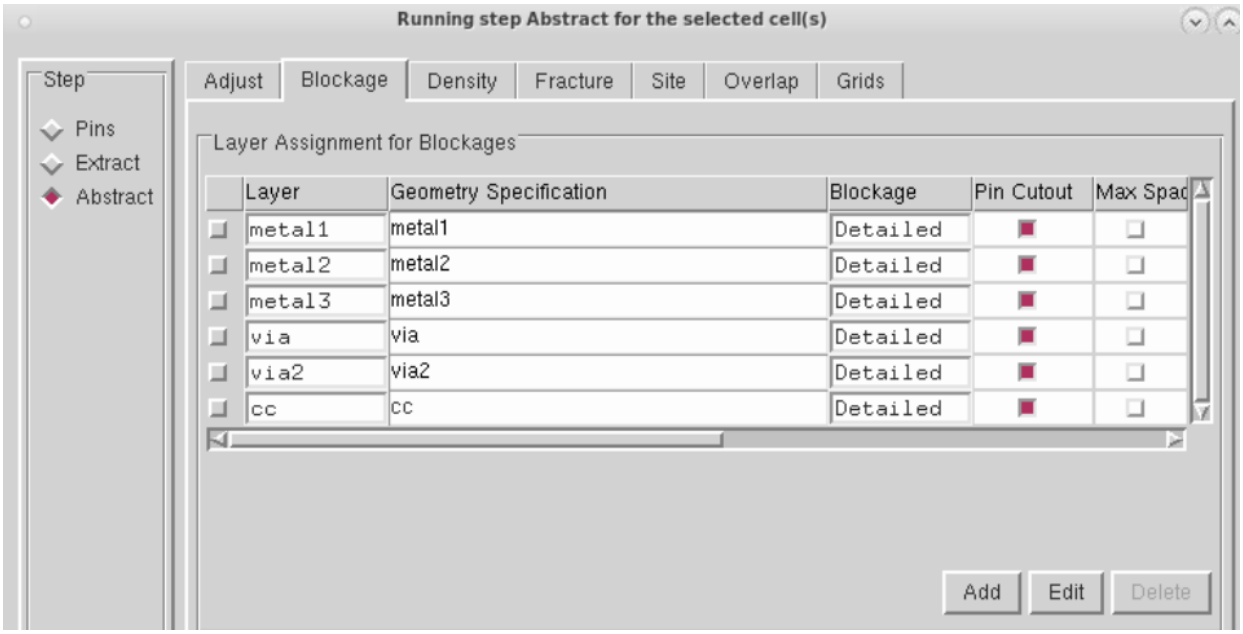


Figure aa: Abstract step with Detailed Blockage and Pin Cutouts specified.

Another small change from the book is that we're now using LEF v5.7 for output.

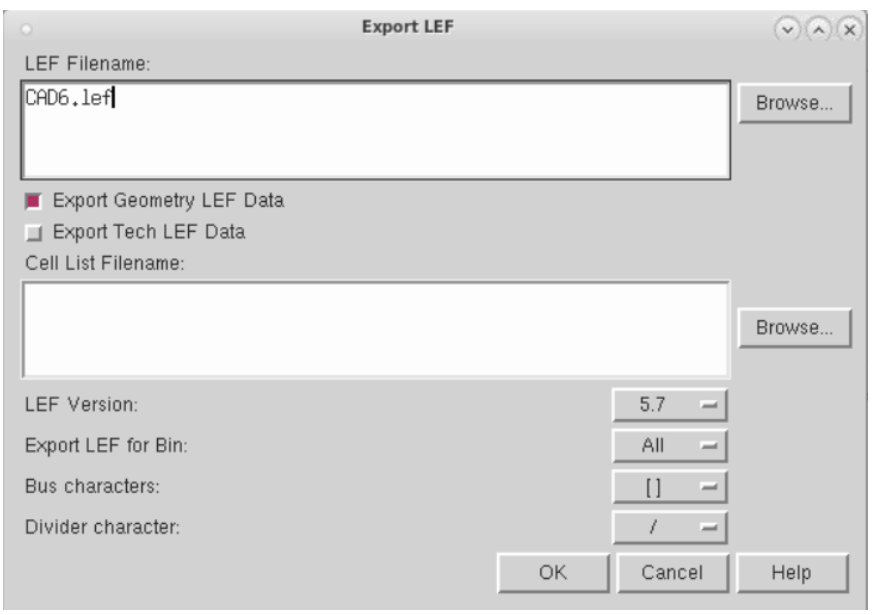


Figure aa: Figure 10.9 from the book. Note that we're now using LEF Version 5.7.

You still need to add the Techheader.lef file to the beginning of the LEF file that is generated by the Abstract tool (Section 10.6). If you did everything correctly, you should be able to look at the abstract view of a cell and see both filled in routing layers (blockages

on those layers), and hollow shapes in the routing layer colors (showing the connected material that can be used to connect to that port of the cell.

Chapter 11: The biggest change with SOC Encounter is that it's now called Encounter Digital Implementation System (EDI). So, the script to use to fire up the tool is now called cad-edi. The other change is that the tool now uses "multi-mode, multi-corner" timing analysis. This means that the tool can make use of much more detailed timing information with multiple timing files being used for different process corners. This means that instead of a single timing file in the configuration, you need to encapsulate timing information into a "timing view" file which we'll call Default.view. After the design configuration stage, things are very similar to the book, but with slightly different looking interfaces in some cases.

When you fire up the tool with cad-edi, the interface looks only a little bit different from the book.

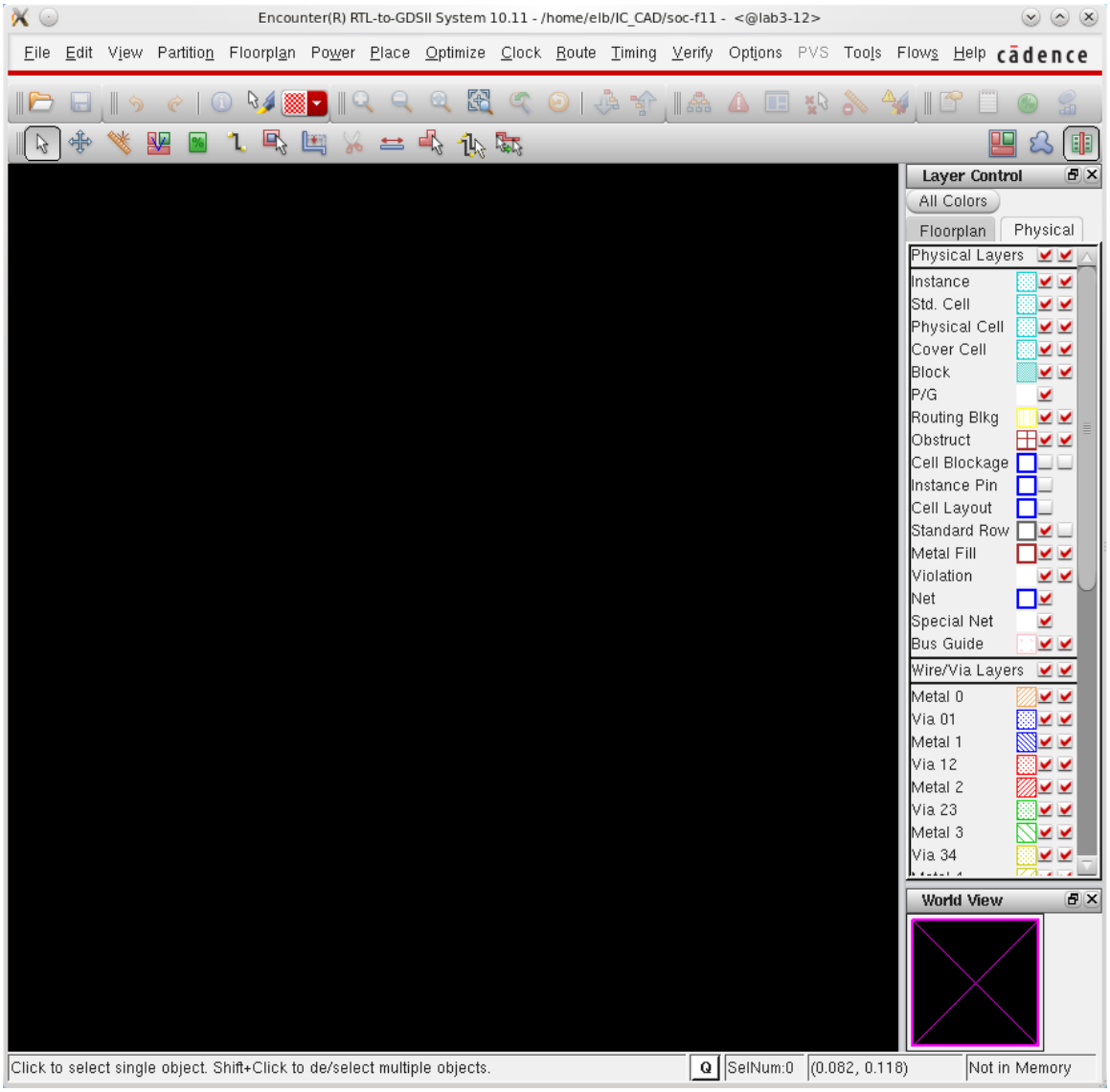


Figure bb: Figure 11.4 from the book.

Before you import your design, you should create two files: <design>.conf and Default.view. The <design>.conf file should be based on the UofU_edi.conf file in the class directory (/uusoc/facility/cad_common/local/class/6710/F11/cadence/EDI/UofU_edi.conf). The things that you need to update are marked with !!...!!. Your Default.view file should also be based on the example in the class EDI directory. Our version o the Default.view file does only simple timing using one timing library and no corner analysis, but that should be fine for class chips.

Once you've created your versions of these two files you can proceed with Design->Import as described in section 11.1.1 in the book. If you've made the <design>.conf file correctly, you can use that to fill in all the fields in the Design->Import form. When you first get the Design->Import form all the fields will be empty. Click on the "Load" button to load your .conf file.

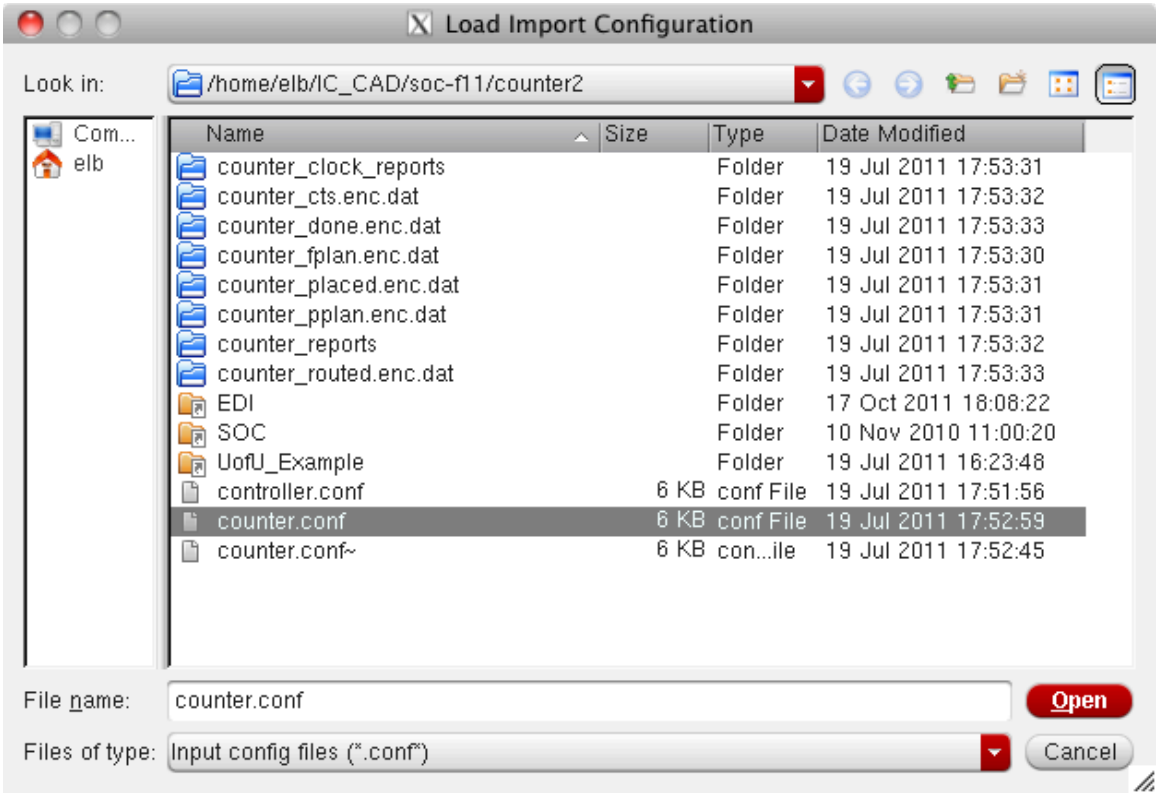


Figure cc: Loading a .conf file for EDI configuration. In this example I'm loading counter.conf .

Once the .conf file has been loaded, you'll see the fields of the Design->Import bx filled in.

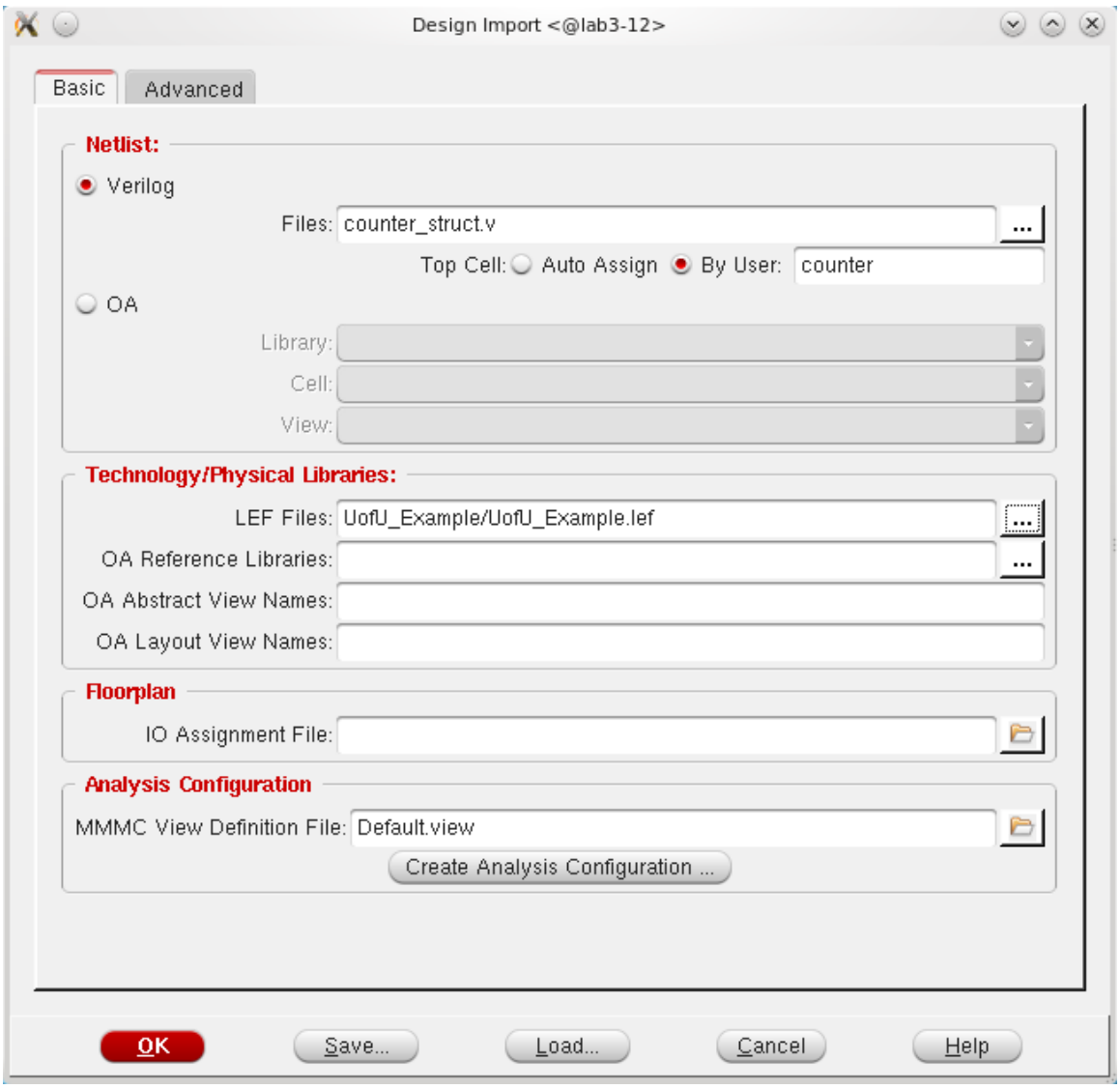


Figure dd: Design->Import dialog box. Note the field for MMMC View Definition File - this is new in this version of EDI.

You may not need to look at this if you have everything set up correctly, but if you want to you can go to the ITO/CTS field under the Advanced tab and notice that the CTS (Clock Tree Synthesis) cell specification has changed. Rather than specify each type of CTS cell separately in terms of their footprints, you simply give the names of all cells you'd like to use in CTS.

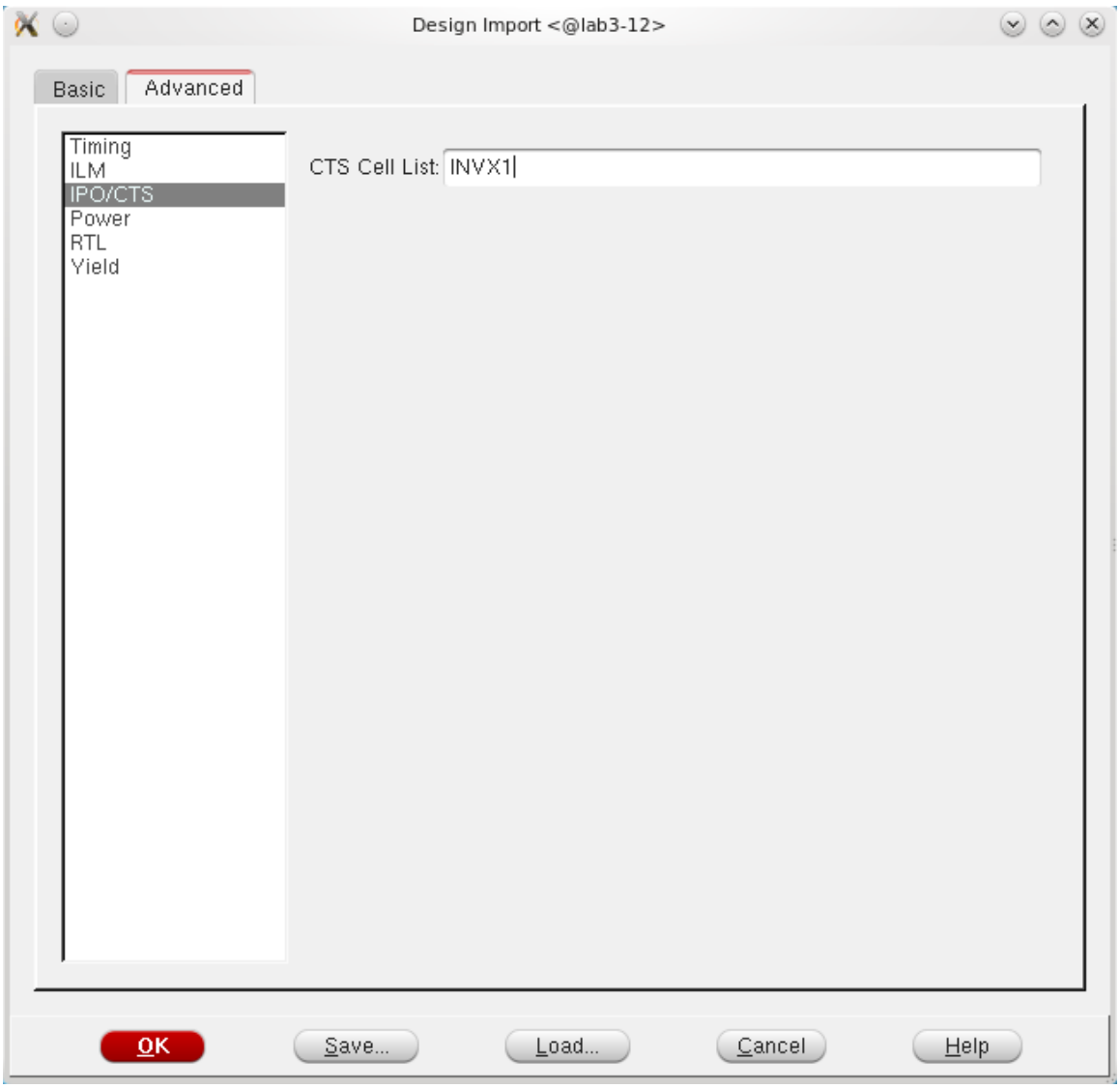


Figure ee: Figure 11.6 in the CAD book - specifying clock tree cells by name.

The next steps in the EDI flow are largely the same as in the book with only very minor differences in the dialog boxes. When you get to placement, you'll notice that the placement includes a "trial route" which is an initial low-quality but fast routing of the circuit to help in the placement. Don't be fooled – you still have to do clock tree synthesis and real routing!

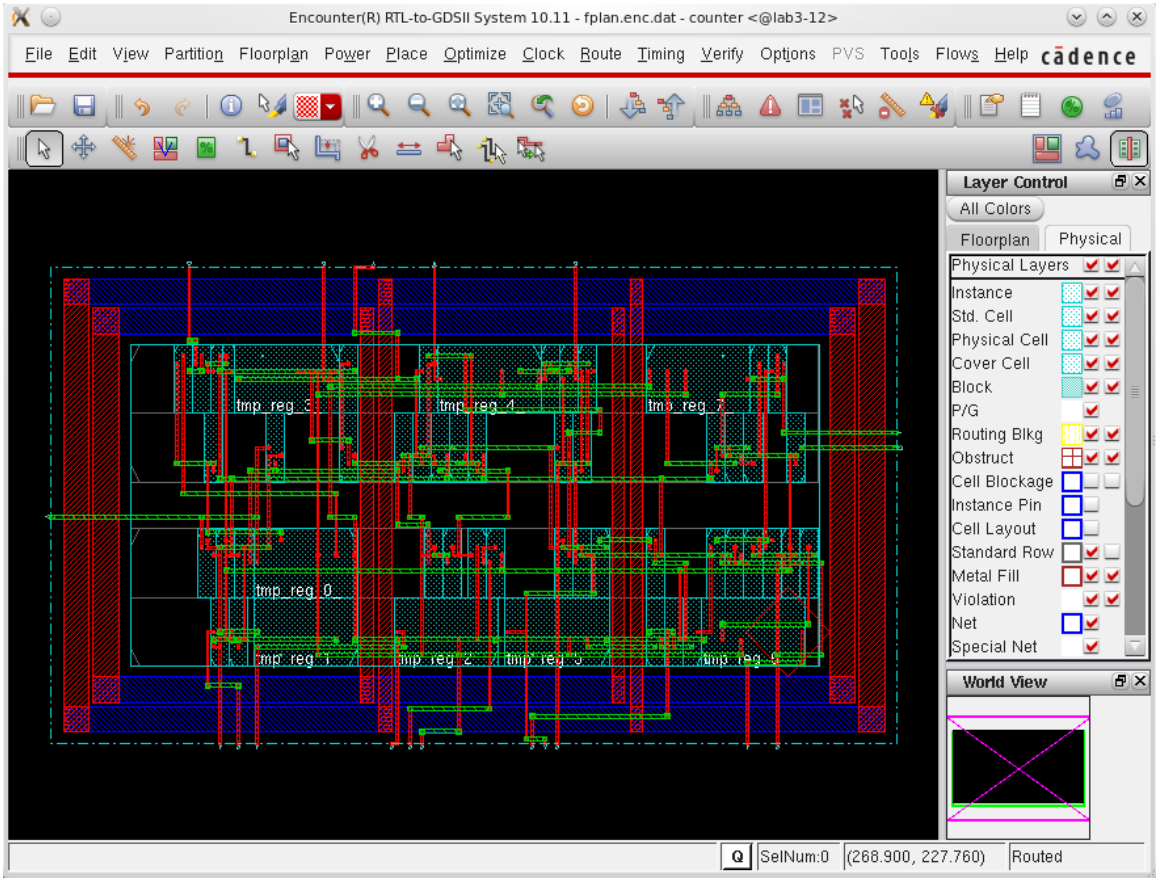


Figure ff: Figure 11.16 in the book showing the circuit after placement Note that trial routing is also shown.

Figure gg: Figure 11.17 in the text - Dialog box for timing optimization

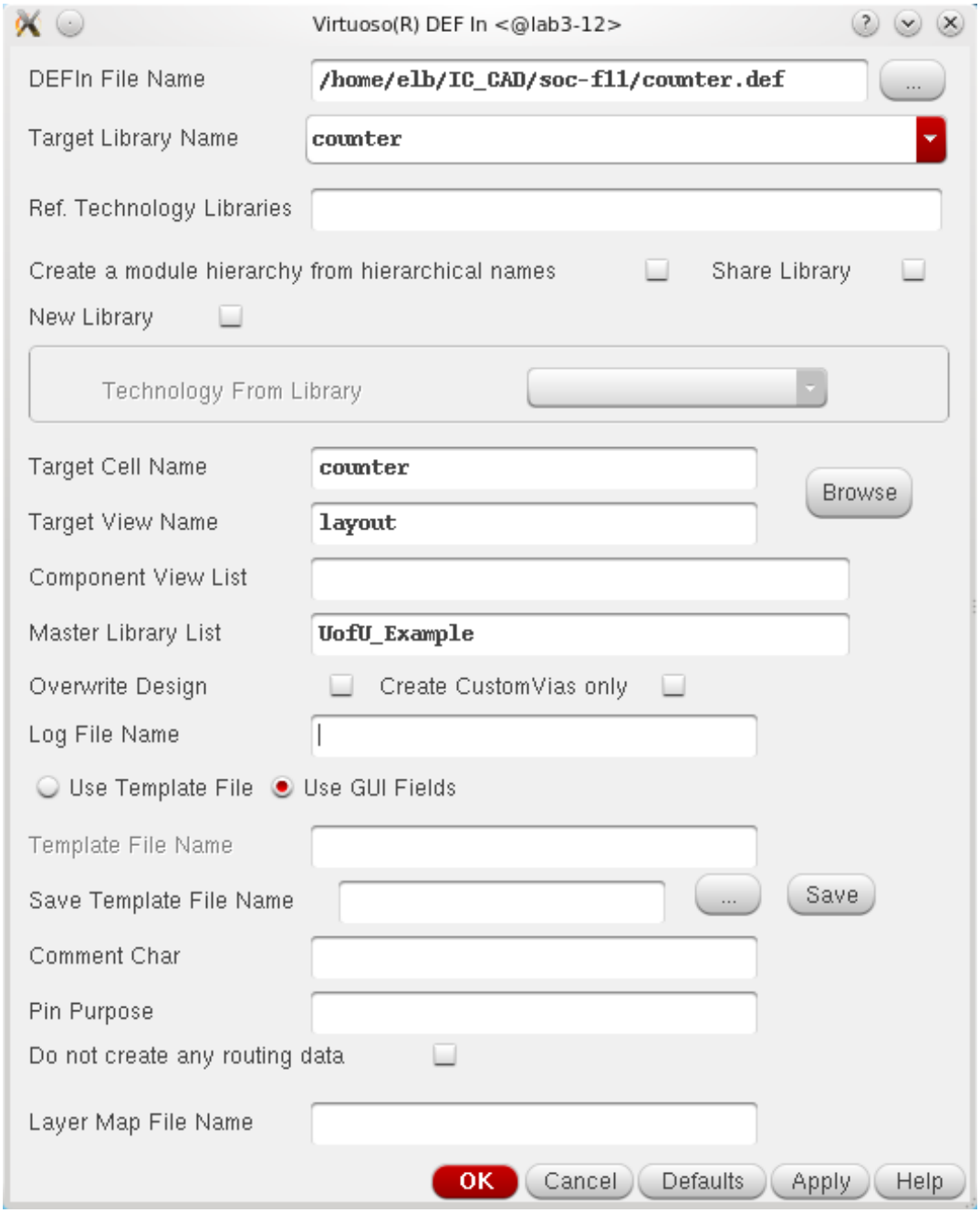
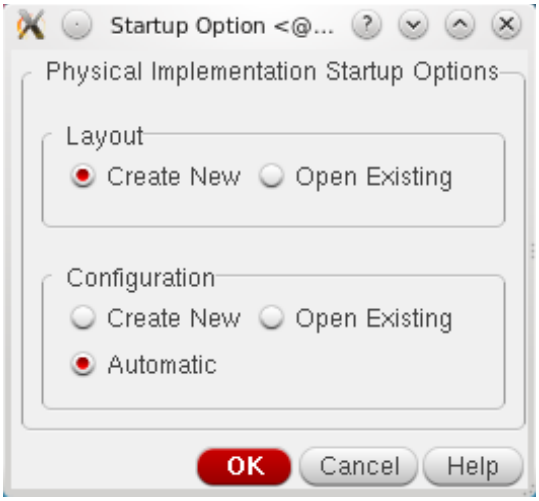


Figure hh: Figure 11.35 in the text - Dialog box for importing DEF files to icfb. This form has some different options than the one seen in the text.

Chapter 12: The chip assembly router s essentially the same as the one called ccar in the text, but is now called vcar (Virtuoso Chip Assembly Router). The interface is largely the same, but has tighter integration with icfb in some ways.

In Section 12.1.1 you start the process by opening the schematic (same as the one seen in Figure 12.1 in the text), and then using Launch->LayoutXL. You'll see a couple dialog boxes along the way.



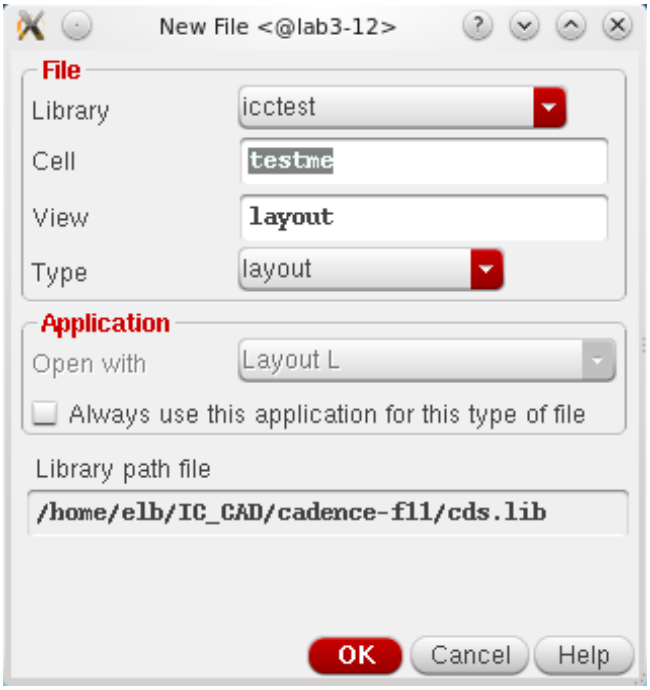


Figure ii: Dialog boxes that appear when you Launch LayoutXL to start the cvar chip routing process. The Startup Option is asking if you want to start with a fresh new layout, and the New File just says that the new view should be a layout view.

Once LayoutXL has opened, you will see different views of both your schematic, and of the new layout view for the generated geometry of the layout. Follow the instructions in the text involving the Design -> GenFromSource but in V6 the command is Connectivity -> Generate -> All From Source, or you can click on the button that looks like this hear the bottom of the window: 

When you do this, you'll get a Generate Layout dialog box. The version in Figure 12.2 in your text has all selections on a single window. The V6 version has separate tabs. The V6 dialog boxes are shown below. For the Generate tab, you can leave the defaults alone. For the I/O pins tab you'll want to update the layer on which you'd like the I/O pins created – I've changed things to metal2 in this example. Also, I prefer to do vdd! and gnd! routing myself, so I select those pins, and then uncheck the "Create" button and then Update (using the controls below the list of pins). You can leave the defaults alone for PR Boundary and Floorplan also.

Figure jj: Generate Layout dialog box - the old box was seen in Figure 12.2 in your text. This V6 version has separate tabs.

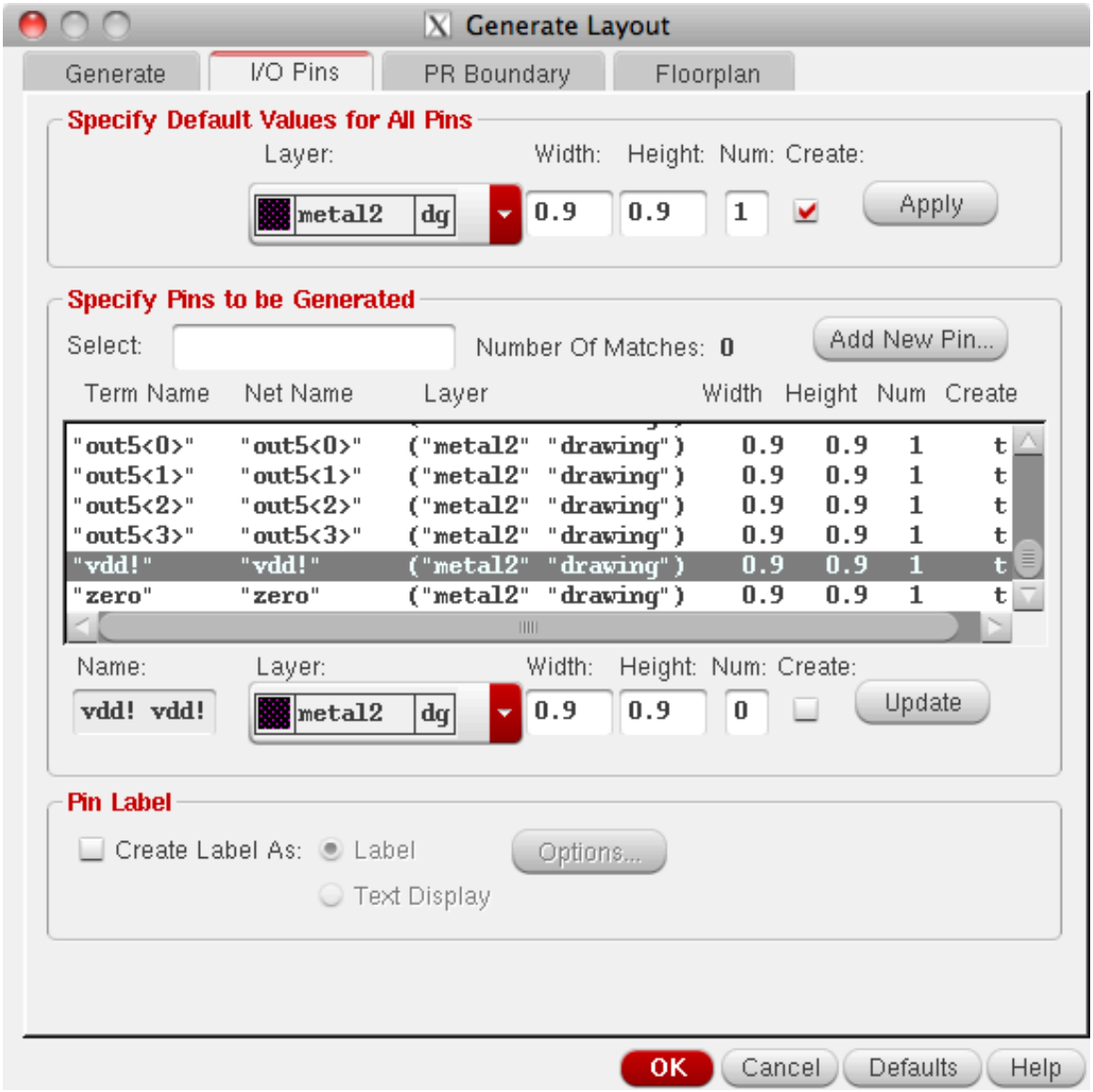


Figure kk: The I/O Pins tab on the Generate Layout box is where you specify what layer you'd like each I/O pin to be on. I like to disable the vdd! and gnd! pins from being automatcally generated. This box shows the state after disabling (un-checking Create) and Updating for vdd!.

After you make your changes to the I/O Pins section and click OK, you'll get the new layout as in Figure 12.3 in the text. The new version looks very much the same. You can also pick up blocks, and pick up I/O pins and move them around as described in the text to get a placement that looks like Figure 12.4. To see the unconnected nets, first analyze the circuit with Connectivity -> Analyze, and then use Connetivity -> Nets -> Show/Hide All Incomplete Nets.

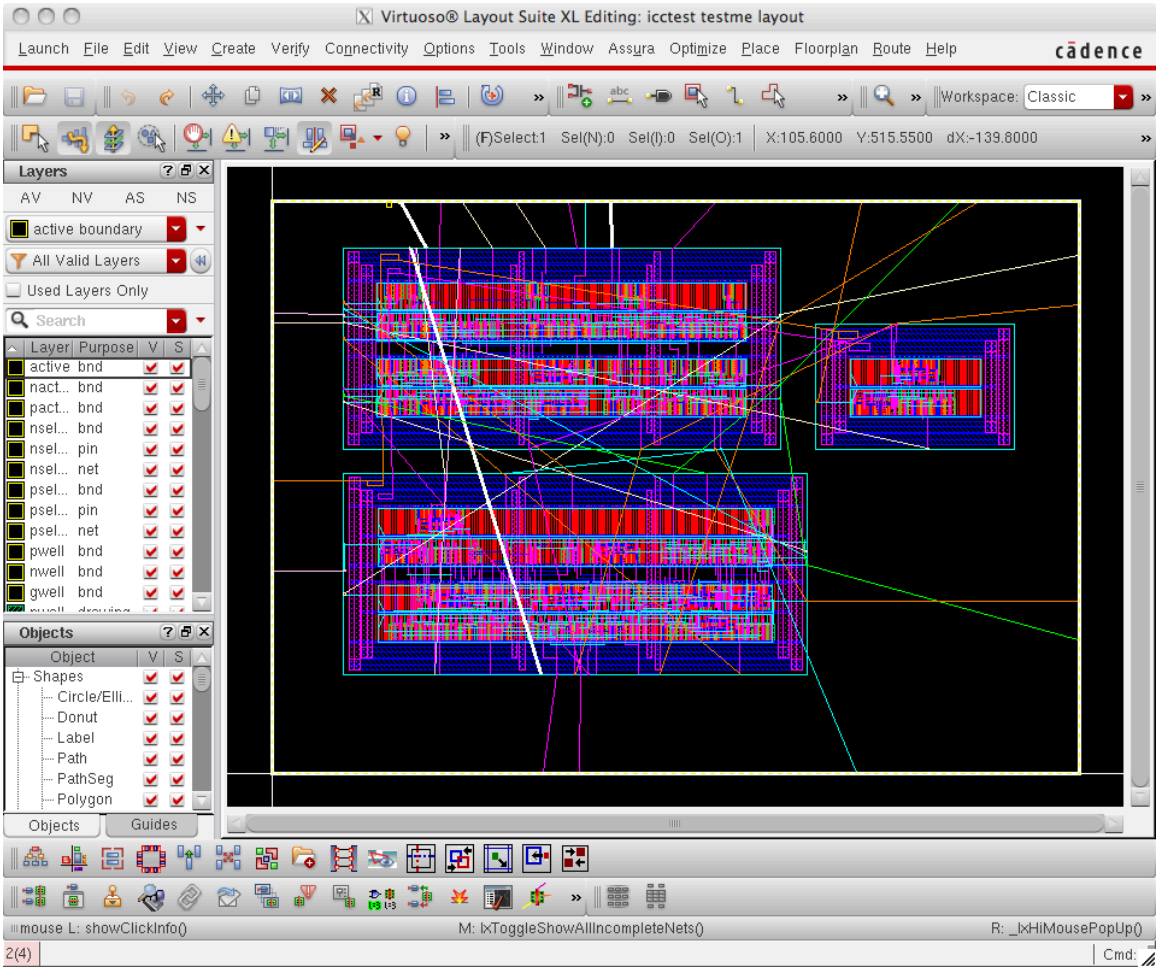


Figure ll: Figure 12.4 in the text, but showing the new V6 look.

Creating the vdd! and gnd! wires that connect the rings of the blocks that came from EDI is the same as described in the text, and shown in Figure 12.5.

Now you can use the procedure in Section 12.1.2 to start up the router. You may get OA (Open Access) warnings about non-via instances. Although I don't know exactly what these warnings are talking about, these appear to be things you can ignore.

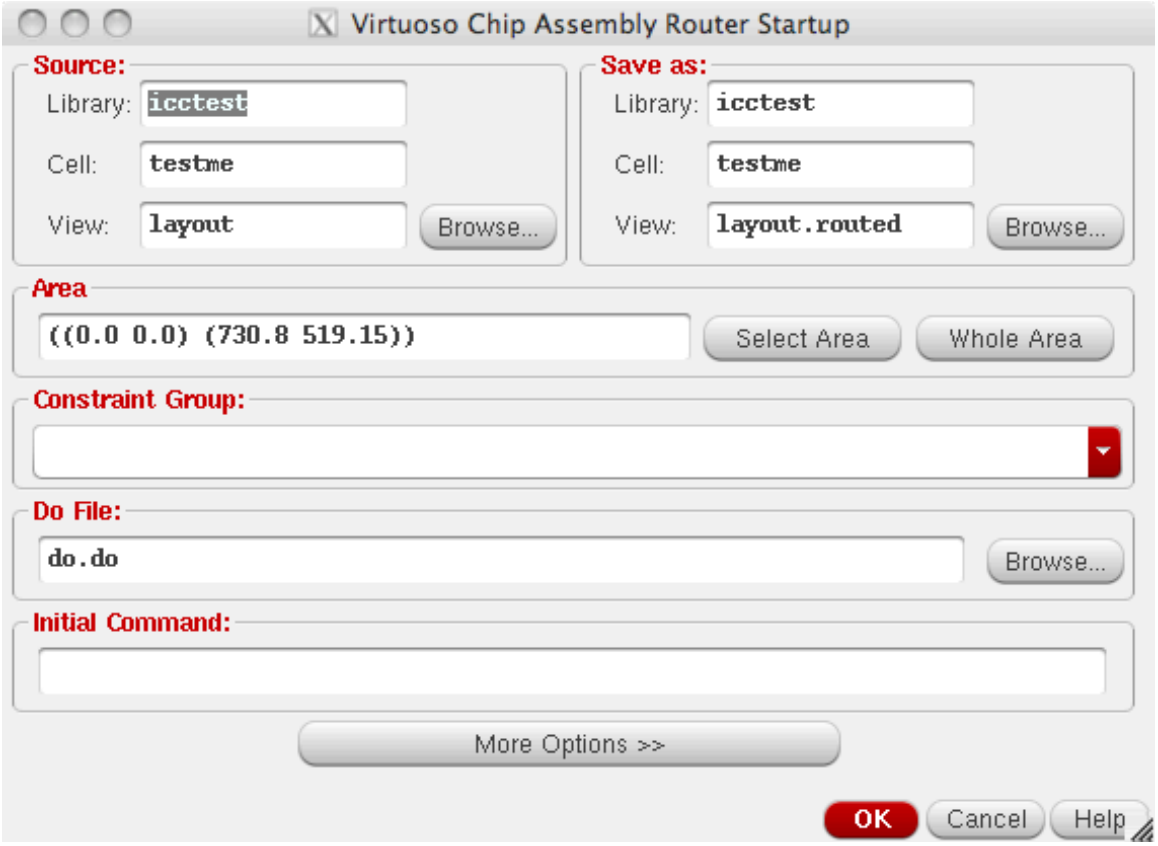


Figure mm: Dialog box for starting the chip assembly router. Nte that it's now called Virtuoso Chip Assembly Router (vcar). Ender do.do into the do file field.

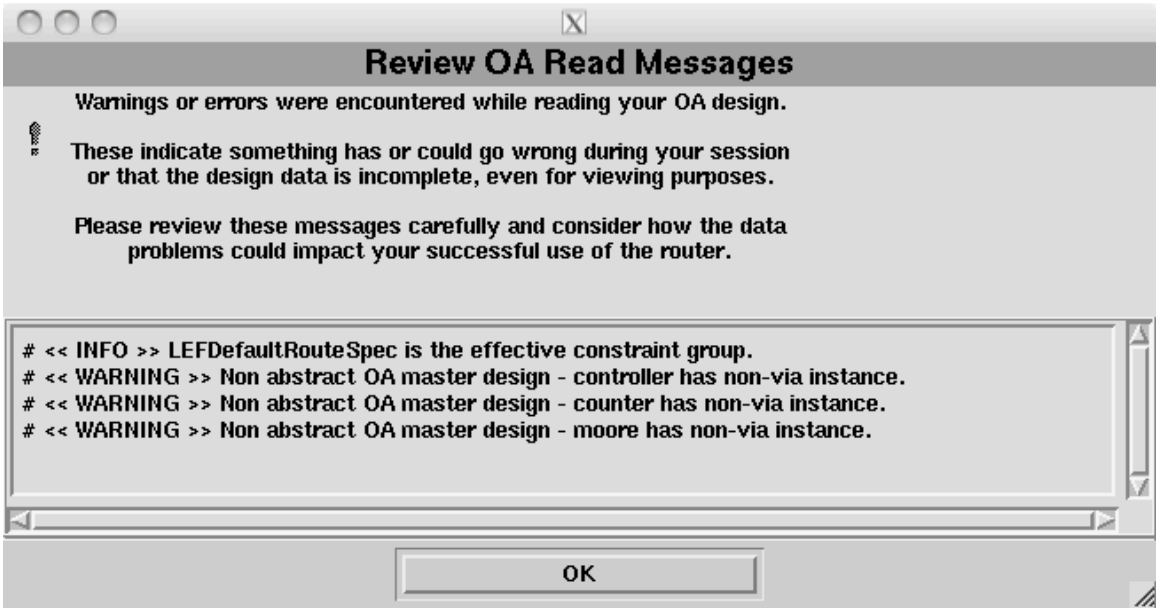


Figure nn: These OA warnings seem to be ignorable.

You can now follow the steps in 21.1.2 to do the routing. In particular, the steps on Page 380 are all still relevant. Note that when you invoke the global router you might get a warning that “this may not be a chip assembly application.” You can ignore this.

IN V6 you no longer need to “write -> session” as described in the text. The layout view in LayoutXL should be updated automatically. Note that it will be updated as a “layout.routed” file instead of a “layout” file.

The rest of Chapter 12, including the pad routing, should follow a similar pattern as in the book, and as modified in the previous examples.

When it comes to generating the GDS (Stream) file, you should use the File->Export->Stream option from the CIW as described in Section 12.3. The Export -> Stream dialog box in V6 now has a Show Options tab that reveals the layer tab. If you Show Options, and then select the Layers tab, you can load a Layer Map File (this is the equivalent of the User-Defined Data described in the text). Load the stream4gds.map file from /uusoc/facility/cad_common/local/class/6710/F11/cadence/map_files as seen in the following figures.

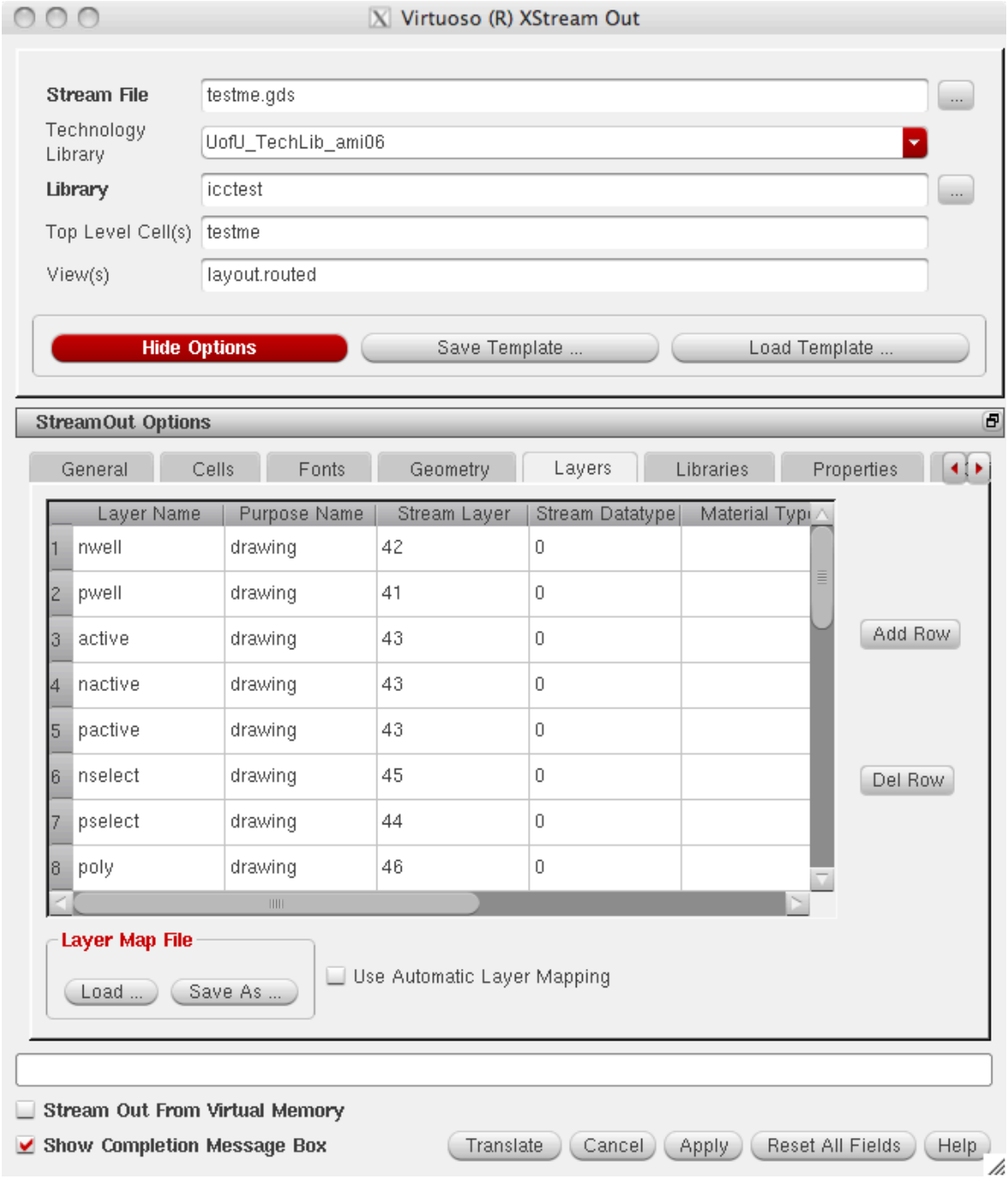


Figure oo: The Export -> Stream dialog box from Figure 12.25 in your text. This is how the box looks after you've loaded the stream4gds.map file. Note that the View is "layout.routed".

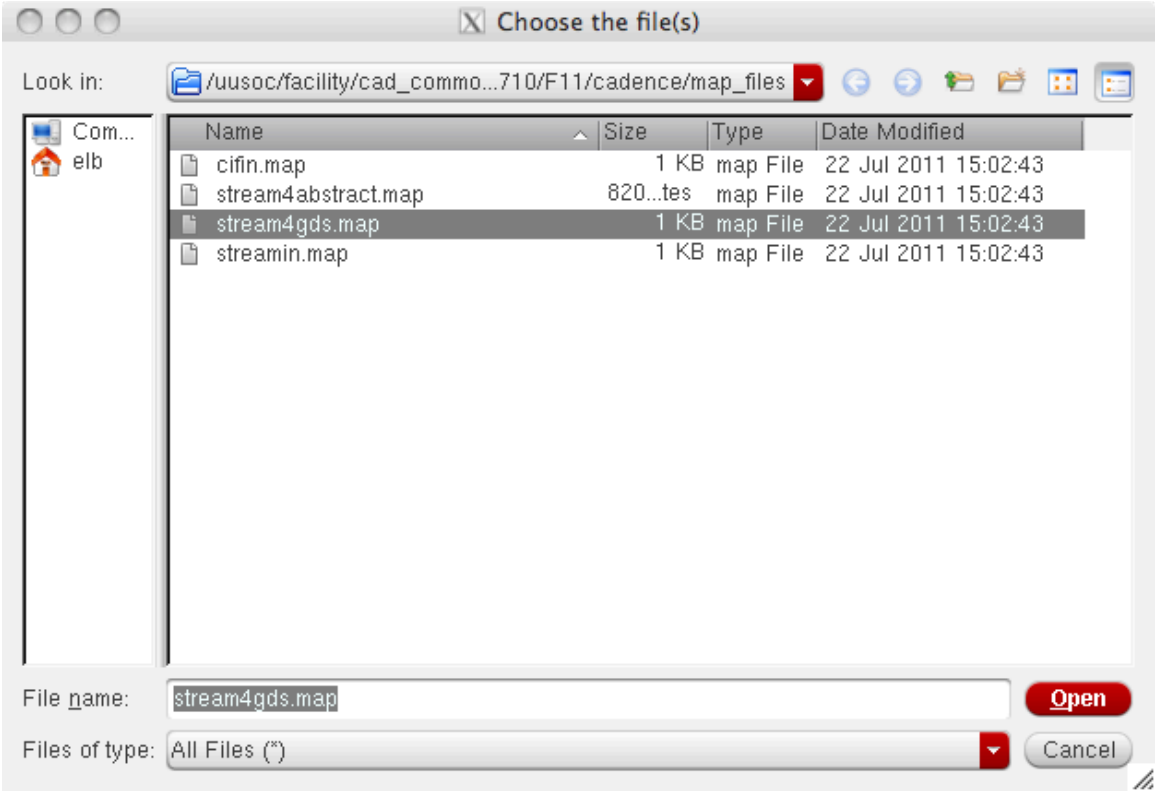


Figure pp: The equivalent of Figure 12.26 in your text. This is how you find the stream4gds.map file to load as a Layer Map File.

After you've exported the gds file, you should read it back in with File -> Import -> Stream from the CIW. You can then DRC and Extract the imported gds information and check to make sure that it's the same as the file you exported. You can use LVS on the extracted file from your layout, and the extracted file from the imported gds to make sure they match.

Stream import requires another map file – this time use the streamin.map file. You should also make a new library to hold the imported information. The import dialog box looks like the following (after the streamin.map has been loaded as the Layer Map File).

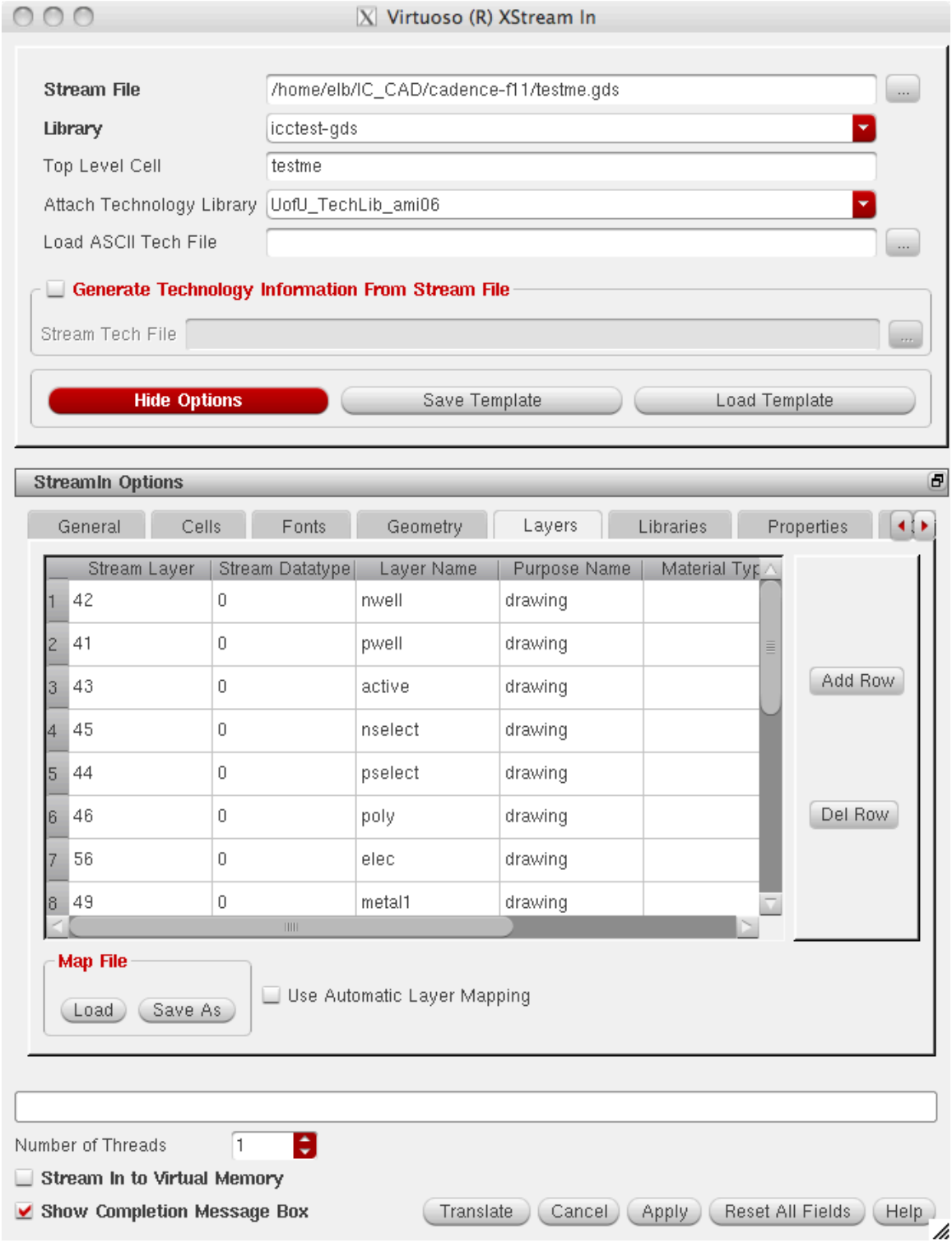


Figure qq: The Import -> Stream dialog box. Note that I've made a new library named icctest-gds to hold this imported information. This box is show after the streamin.map Map File has been loaded.

Chapter 13: The content of Chapter 13 doesn't really change. Some of the steps are different than in the book, but those modified steps are documented in the respective chapters, so they're not repeated here.

New Material: Hierarchical Layout Place and Route Flows

Here is some documentation on taking a previously placed and routed circuit and making is usable as a macro cell in EDI. That is, using EDI as a chip assembly router instead of vcar. DISCLAIMER: The instructions listed here worked for the V5 tools. I haven't gone through in detail and checked everything for the V6 flow. There may be slight differences in the V6 EDI flow that you'll have to navigate.

Procedure for EDI/VCAR hierarchical flow

For this flow you create pieces of the design in EDI. Those pieces will be self-contained layout (DEF can be read back in to icfb to become layout views) with power and ground rings.

These, and other custom blocks, can be placed by hand, power and ground routed by hand, and then the signals can be connected in ICC VCAR (Virtuoso Chip Assembly Router) as described in the CAD manual and in the previous V6 discussions in this document. See Chapter 12 for more details about VCAR.

Procedure for hierarchical EDI flow

You might also want to take blocks that are placed and routed in EDI, and use them as macro blocks in a hierarchical use of EDI. That is, the blocks would be placed within another EDI floorplan and be part of the EDI-style power/ground grid, and standard cells would be placed around the macro blocks. This is possible too. Note that the easy way involves having self-contained power and ground rings around the macro blocks anyway, but then those rings will be connected into the global power/ground structure of the EDI floorplan.

Step one is preparing your circuits as hierarchical blocks. These are blocks that you want to use in another EDI run. They might be macro blocks, or it might be the whole core. To do this you'll eventually need a .lib description of the cell for timing, and a .lef version of the cell for physical place and route. You can get the .lib from EDI, but you'll have to generate the .lef through abstract which means importing the .def into icfb and then running abstract.

1 - place and route your block with EDI. Make sure to put a vdd/gnd ring around your block. You might be able to make this work without one, but I know it works with one. You can use stripes if you like too, but they are not required.

2 - When you have finished and have a correctly placed and routed circuit you export a structural verilog file and def file as usual. You should also generate a .lib file These steps are included in the “verify.tcl” file that is part of the general CS6710 EDI scripted flow.

```
puts "----------Output ${BASENAME}.def file------------"
# Export the DEF, v, spef, sdf, lef, and lib files
global dbgLefDefOutVersion
set dbgLefDefOutVersion 5.6
defOut -floorplan -netlist -routing $BASENAME.def
saveDesign ${BASENAME}_done.enc -def

puts "----------Output ${BASENAME}_soc.v file---------"
SaveNetlist [format "%s_soc.v" $BASENAME]

# generate final timing data
extractRC -outfile $BASENAME.cap
rcOut -spef $BASENAME.spef
write_sdf -ideal_clock_network $BASENAME.sdf

# generate a .lib model of the cell just in case
do_extract_model -blackbox_2d  ${BASENAME}_soc.lib

# Generate timing model for PrimeTime just in case
writeTimingCon ${BASENAME}_done.pt
report_timing -check_clocks report_timing.pt

puts "------------Verify and file output done------"
```

4 - Now you have the .lib. If you want to use this cell as an instance in Synopsys dc_shell synthesis and you don't want Synopsys to complain about not knowing what it is (this complaint can be ignored, but it is annoying) you can compile the .lib into a .db file so the synthesis process can use it. Use dc_shell read_lib and write_lib for this.

5 - Import the .def file into icfb using import -> def. Go through the regular abstract view to layout view stuff to get a good layout view. Run DRC and Extract.

6 - Import the structural Verilog from EDI into the same library as the layout.

7 - Compare the layout (extracted) and schematics with LVS to make sure things are right.

8 - The abstract process needs to know where vdd! and gnd! are so put shape pins on the vdd! and gnd! wires in the power ring. The I/O signal pins will all be there from EDI.

9 - Run Abstract to generate the abstract view, and the .lef file. You can run abstract from inside composer or from the command line. The important stuff is:

Pins Step:

- The Boundary Adjust is 0. That is, the prBoundary is right on the edge of the cell.

- Make sure that "create power pins from routing" is set and uses layers metal1, metal2, via.

Extract Step:

- Make sure "Extract Power Pins" is enabled

Abstract Step:

- Select "Create Ring Pins" so that the power pins in the .lef file will be "class ring" This tells EDI that there's a power ring to connect to.

- Select "Create Boundary Pins" too so that you'll get I/O pins.

- Blockage tab - cover type, Boundary should be offset so that the cover obstructions in the lef won't extend all the way to the edge of the cell and keep power routing from getting inside the cell. The amount depends on how much space you've used for the rings in your design. If you left a 30 micron gap and filled that with the power and ground rings, then something close to 30 is good. I use 27 in that case.

10 - When you have a successful abstract view, output a .lef file.

11 - (optional) edit this lef file to take out the rotation symmetry from the macro description. If you leave this in, EDI might rotate your cell. I find that if you rotate it things get ugly because you've changed the normal h/v routing conventions. So, take the following lines and modify them by removing the R90. It's fine to flip a macro in X or Y, but rotating seems to cause problems.

```
MACRO macro_name
   CLASS RING ;
   FOREIGN macro_name 0 0 ;
   ORIGIN 0.00 0.00 ;
   SIZE 866.10 BY 843.00 ;
   SYMMETRY X Y R90;
   ...
```

You now have a macro with layout, extracted, schematic, symbol, and abstract views. You also have .lib, .lef, .v, and possibly .db descriptions of the cell. It's now ready to use in EDI as a fixed block in another piece of Verilog code.

Step 2 is to use the block in another macro. The easy part is putting it in your code. You can put it in a piece of structural verilog by hand, or you can include that instance in a larger piece of Verilog that has behavioral stuff and pass the whole thing through Synopsys synthesis to get a purely structural file.

Remember that if you run Synopsys on it you will need the .db description so that Synopsys doesn't complain. Actually you can probably ignore the complaints but it makes it nicer not to see the complaints. This .db file would be listed along with the other .db files in the target library list.

When used as in instance you should always use "dot notation" for the arguments to the instance. That is, don't use:

blockname ID (A, B);

Instead use

blockname ID (.arg1(A), .arg2(B));

With these "black boxes" the tools may or may not have info about the order of arguments in the module so this eliminates that worry.

Now I assume that you have a sturctural Verilog description that has instances of your block or blocks in it. For example:

```
module alu_core ( result, a, b, reset, clk, func );
  output [15:0] result;
  input [7:0] a;
  input [7:0] b;
  input [3:0] func;
  input reset, clk;
  wire   n2, n3, n4, n5, n6, n7, n8, n9, n13, n15, n17, n19, n21, n23, n24,
         n261, n262, n263, n264, n265;
  wire   [7:0] maca;
  wire   [15:0] product;
  wire   [7:0] macb;
  wire   [8:0] sum;
  wire   [15:0] aluout;

// instances of pre-designed modules
  mult Umult ( .P(product), .A(a), .B(b) );
  add Uadd ( .S(sum), .A({maca[7], n264, maca[5:0]}), .B(macb) );

// Standard cells that need to be placed and routed with EDI
  DFF result_reg_15_ ( .D(aluout[15]), .G(clk), .CLR(reset), .Q(result[15]) );
  DFF result_reg_14_ ( .D(aluout[14]), .G(clk), .CLR(reset), .Q(result[14]) );
  DFF result_reg_13_ ( .D(aluout[13]), .G(clk), .CLR(reset), .Q(result[13]) );
```

```
  DFF result_reg_12_ ( .D(aluout[12]), .G(clk), .CLR(reset), .Q(result[12]) );
  DFF result_reg_11_ ( .D(aluout[11]), .G(clk), .CLR(reset), .Q(result[11]) );
  DFF result_reg_10_ ( .D(aluout[10]), .G(clk), .CLR(reset), .Q(result[10]) );
  MUX2_INV U30 ( .A(b[7]), .B(result[7]), .S(func[3]), .Y(n2) );
  MUX2_INV U31 ( .A(b[6]), .B(result[6]), .S(func[3]), .Y(n3) );
  NAND2 U46 ( .A(n24), .B(n25), .Y(aluout[9]) );
  NAND2 U47 ( .A(n210), .B(product[9]), .Y(n25) );
  NOR2 U87 ( .A(n213), .B(n74), .Y(n73) );
  MUX2_INV U88 ( .A(n43), .B(n44), .S(a[5]), .Y(n74) );
  NAND2 U89 ( .A(n263), .B(a[5]), .Y(n72) );
  NAND2 U91 ( .A(n211), .B(n13), .Y(n76) );
  NAND2 U92 ( .A(n256), .B(n194), .Y(n75) );
  ...
endmodule
```

Now you can generate a new placed and routed module that includes your blocks and (perhaps) standard cells that should be placed around the blocks. This new block can be made into yet another macro by following the procedure above to generate .lib, .lef, .v, and .db descriptions.

1 - edit the config file to make sure that all the .lib files are included in the timing librarys (standard-cells.lib, block.lib, etc.).

2 - Ditto for the lef files (techheader.lef, cells.lef, block.lef, etc.)

3 - Other config stuff is the same as before - the cell name, input netlist (.v), timing constraint file (sdc) etc. are set as before. You may choose to include a .io file to specify pin placements, or you can leave that out.

4 - Add a couple steps to the normal flow to deal with the blocks.

- After floorplan sizing, you should add a halo around each block so that standard cells aren't placed too close to the block. It doesn't need to be much, but it should be some. The following will add 3 microns of halo to everything in class block. You can do the same from the menu using Floorplan-EditFloorplan-EditHalos. There are two types - routing halos that restrict routing in that halo, and placement halos that restrict cell placement.

```
addHaloToBlock 3 3 3 3 –allBlock
addRoutingHalo -space 3 -top M3 -bottom M1 -allBlocks
```

- Once the block have halos you can place them in the core routing area. You can do this by hand (pick up and move them in the gui with the movement widget), or have EDI do an auto placement for you. The gui is Floorplan-AutomaticFloorplan-PlaceDesign. This will place all the blocks for you. The script version of the command is

planDesign

You can also place things relative to other blocks or sides of the core using the Floorplan-RelativeFloorplan dialogs (start with EditConstrants or GenerateConstraints).

- Once the blocks are placed you need to set their placement status to fixed so that standard cell placement won't change them. You do this with Floorplan-EditFloorplan-SetBlockPlacementStatus or with the script command:

```
setBlockPlacementStatus -allHardMacros -status fixed
```

- Now you can make the power rings and stripes as usual. The extra settings you need to be careful of are:

- Add Rings - Same as for flat design

- Add Stripe - Set Configuration, Set Patten, Stripe Boundary, and First/Last are the same as in flat. In Advanced make sure Snap Wire Center To Routing Grid is set to Grid. In Via Generation select If Same Layer Target Exists... to be Same Layer Target Only. This keeps problems from happening later with overlapped vias at the corners of the block power rings.

The script commands are (assuming that $pspace, $pwidth, $sspace, and $soffset have been set in the script):

```
addRing -spacing_bottom $pspace -width_left $pwidth -width_bottom $pwidth
    -width_top $pwidth -spacing_top $pspace -layer_bottom metal1 -center 1
    -stacked_via_top_layer metal3 -width_right $pwidth -around core -jog_distance
     $pspace -offset_bottom $pspace -layer_top metal1 -threshold $pspace
    -offset_left $pspace -spacing_right $pspace -spacing_left $pspace -offset_right
    $pspace -offset_top $pspace -layer_right metal2 -nets {gnd! vdd! }
    -stacked_via_bottom_layer metal1 -layer_left metal2

addStripe -same_layer_target_only 1 -block_ring_top_layer_limit metal3
    -max_same_layer_jog_length 3.0 -snap_wire_center_to_grid Grid
    -padcore_ring_bottom_layer_limit metal1 -set_to_set_distance $sspace
    -stacked_via_top_layer metal3 -padcore_ring_top_layer_limit metal3
    -spacing $pspace -xleft_offset $soffset -merge_stripes_value 1.5 -layer metal2
    -block_ring_bottom_layer_limit metal1 -width $swidth -nets {gnd! vdd! }
   -stacked_via_bottom_layer metal1
```

- Now you have the blocks placed and the rings and cores placed. You should now place the cells before you sroute the power and ground lines. I've seen cases where there are gaps in the power lines if you don't place the cells first.

- When you do sroute the power and ground lines you should add the option in the Via Generation tab to Connect to Overlapping Targets on the Closest Layer. This seems to help with the overlapping via problem. The text command in the script is:

sroute -viaThruToClosestRing -jogControl { preferWithChanges differentLayer }

- Other steps should be the same as in the flat flow.

An alternative is to do things in a slightly different order and take advantage of the "place blocks and cells" command to place them all in one shot.

- In this flow you set the floorplan size in the usual way. You can also make the power ring as usual.

- Make halos around the blocks first before you place anything.

- Now, you can place the cells and blocks in one step using the Place-PlaceStandardCellsAndBlocks command. Script is:

```
setPlaceMode -timingdriven -reorderScan –congMediumEffort
   -noCongOpt -noModulePlan
placeDesign -inPlaceOpt -prePlaceOpt
```

- Now you can make the power stripes. Note that the stripes will connect to the ring connections of the blocks. But, they also splat right over the standard cells!

- So, now you need to refine the placement to move cells out from under the stripes. This is Place-RefinePlacement. It will move the cells so they're not overlapping with the stripes. Script step is:

```
refinePlace
```

- Now you can sroute the power and ground. Remember to use the Overlapping Targets - Closest Layer option.

```
sroute -viaThruToClosestRing -jogControl { preferWithChanges differentLayer }
```

- Now you should have a legal placement and power plan so you can continue with the rest of the standard flow...

Note that if you're running EDI on a cell with pads then:

- You MUST have a .io file with pad placements in it.

- The pads defined in the .io file must be defined in the structural Verilog file and must have the same names.

- When you get to the SpecifyFloorplan step you need to make sure that you don't specify a floorplan that will change the pad cell spacing! Instead of specifying Size By Core Size and Aspect Ratio, you specify the Size By Die Size and make sure that the die size is exactly

the right dimensions. For a single TCU this is 1500 by 1500 for Die Width and Size. Then you can adjust the Core to IO Boundary numbers to change how much space is between the core and the pad ring. The script version of the command uses the -d switch to specify the die size instead of the -r switch for relative aspect ratio and density. An example for a single TCU with 60 microns between the core area and the pag ring (for the power rings) is:

```
floorPlan -site core -d 1500.0 1500.0 60 60 60 60
```