Chapter 9

Abstract Generation

IN ORDER for the place and route tool to do its job, it needs to know certain physical information about the cells in the library it is using. For example, it needs to know the bounding box of each cell so it can use that bounding box when it places cells next to each other. Note that this bounding box may be described by the furthest pieces of geometry in the layout view of the cell, or it may be offset from the layout so that the cell geometry overlaps or has a gap when it is placed next to another cell. The place and route tool doesn't need to know anything about the layout. It just places the cells according to their bounding box.

The other critical information for the place and route tool is the signal connection points on the cells. It needs to know where the power and ground connections are, and the signal connections so that it can wire the cells together. It also needs to know just enough about the layout geometry of the cells so that it know where it can and can't place wires over the top of the cell.

A view of the cell that has only this information about the cell (bounding box, signal connections, and wiring blockages) is called an **abstract** view. An **abstract** view has much less information than the full layout view which is useful for two reasons: it's smaller and therefore less trouble to read, and it doesn't have full layout information so cell library vendors can easily release their abstract views while keeping the customers from seeing the full layout views and thus retain their proprietary layout.

As an example, consider a simple inverter. Figure 9.1 shows the **layout** view side by side with the **abstract** view. You can see that the abstract is very simple compared to the layout. The abstract view doesn't need any information about any layers that aren't directly used by the place and route program. Because the place and route program uses only the metal layers for routing, only the metal layers show up in the abstract. This example

actually has four connection points: A (input), Y (output), and vdd (the top metal piece) and gnd (the bottom metal piece). Note that the power and ground connections are designed to abut (actually overlap) when the cells are placed next to each other with their bounding boxes touching.

9.1 Abstract Tool

We, of course, have full layout views because that's what we've been designing. The tool that extracts the **abstract** information from the full layout view is called, appropriately enough, **abstract** and it can be run from your cadence directory using the `cad-abstract` script.

Before you run **abstract** you need to have **layout** and **extracted** views (at least) of the cells that you wish to generate the abstracts of. Connect to the directory from which you run Cadence, and fire up the tool with `cad-abstract`. The **abstract** tool will now have access to your design libraries that you've been using in Cadence.

9.1.1 Reading your Library into Abstract

You can also get short help on many things by hovering your mouse over the object.

When you start up **abstract** you'll get a command window. All the major operations in generating abstracts can be initiated with the widgets near the top of the window or by using the menus. The far-left widget that looks like a book is used to load a Cadence library into **abstract**. The first step in generating **abstract** views is to load a library that contains your cells. Figure 9.2 shows the **Open Library** box where I'm choosing my **abs-test** library.

Once you've loaded your library you should see all the cell names in your library sorted into **bins**. the **Core** bin will contain the cells that **abstract** thinks will be general core cells. The **Ignore** bin are the cells that it will ignore. The other bins (**IO, Corner, Block**) are related to pad I/O and pre-designed blocks. We won't use them for our cell library. If you are only interested in generating abstracts for some of your cells you can use the **Cells** → **Move...** menu choice to move cells to different **bins**. By moving cells to the **Ignore bin**, for example, you can exclude them from further processing. My example library has 11 cells that I would like to process into **abstract** views, and 2 that I want to ignore. The window looks like that in Figure 9.3. Note that you may have warnings that you don't already have **abstract** views. You can ignore them. You should, however, see green check marks by each cell in the **layout** column to indicate that the layout has been successfully read. If you see an orange exclamation point instead that means that there was some issue when the cell was read into **abstract**.

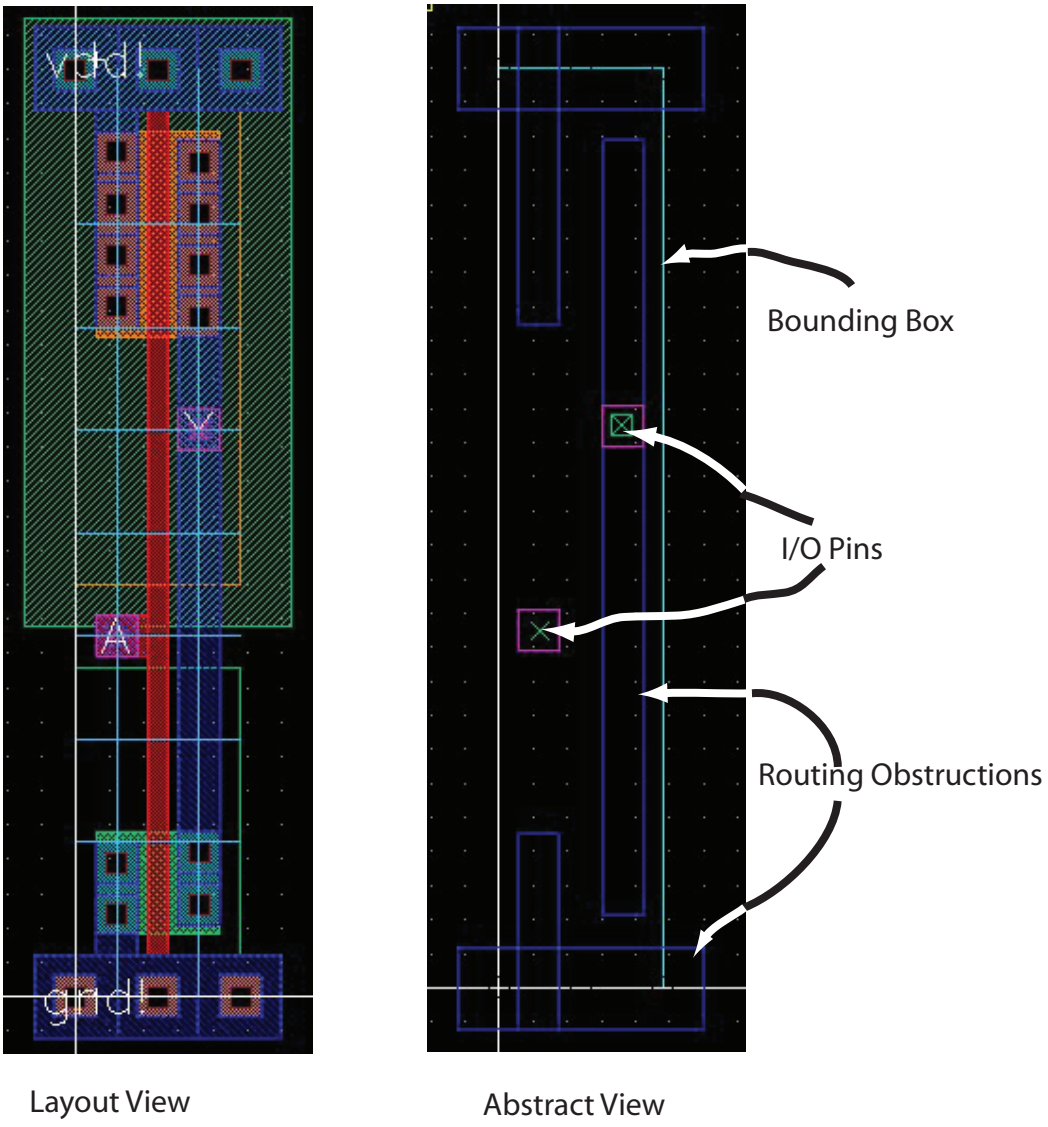

Figure 9.1: Side by side **layout** and **abstract** views of an inverter

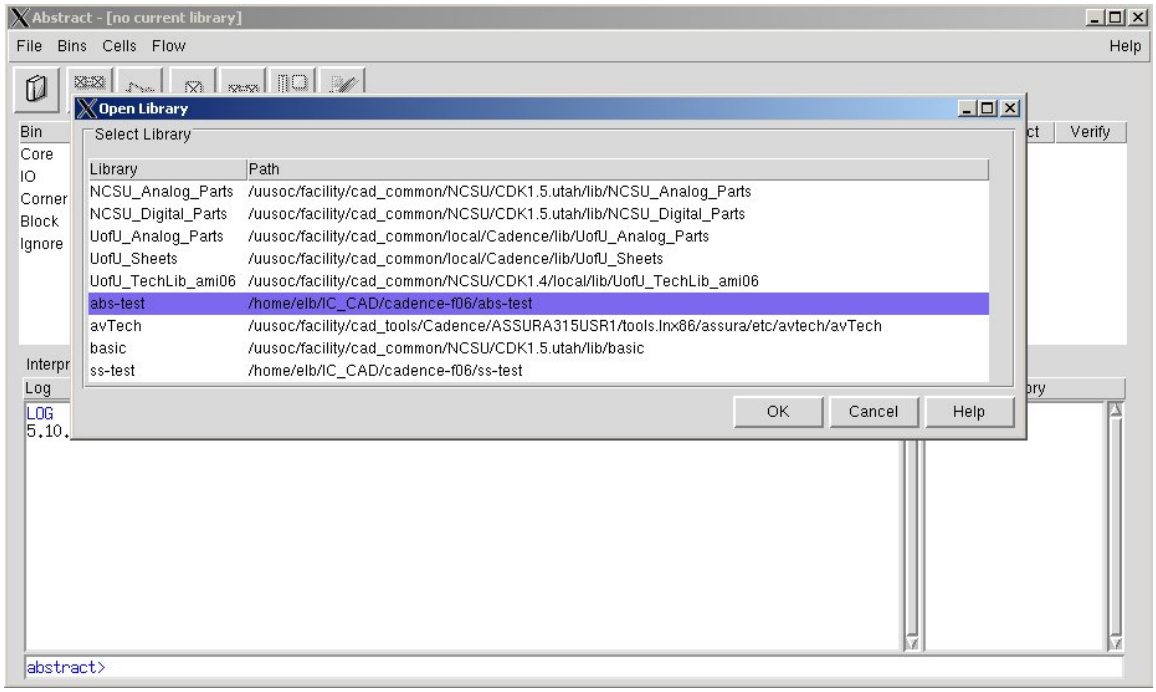

Figure 9.2: Opening a library in **abstract**

You need to look at the log to figure out what's going on. Remember that warnings may often be ignored, but you need to know why you can ignore them! Errors are something you need to fix.

9.1.2 Finding Pins in your cells

Once you've read in your library and you have green checks by all the cells that you want to convert, select the cells you're interested in by clicking and dragging to highlight them, and move to the next step, which in our case is the **Pins** step. We skip the **Logical** step because our place and route tool don't care about the logical behavior of the cells. We put that information in the **.lib** file. The **Pins** step can be initiated by the menu (**Flow** → **Pins...**) or by the **Pins** widget which is the single square with the cross in it (looks like a contact). Initiating the **Pins** phase will bring up the **Pins** dialog box as seen in Figure 9.4. In the **Pins** dialog box you need to add some information.

1. If you have made *all* your pins as **shape pins** as described in Chapter 5, Section 5.2, then you can leave the **Map text labels to pins:** section blank. If you have made pins by just putting a text label on top of some routing material then you need to add something here.

*The notation (**text drawing**) is a layer-purpose pair which means the rectangle is **text** type with **drawing** purpose.*

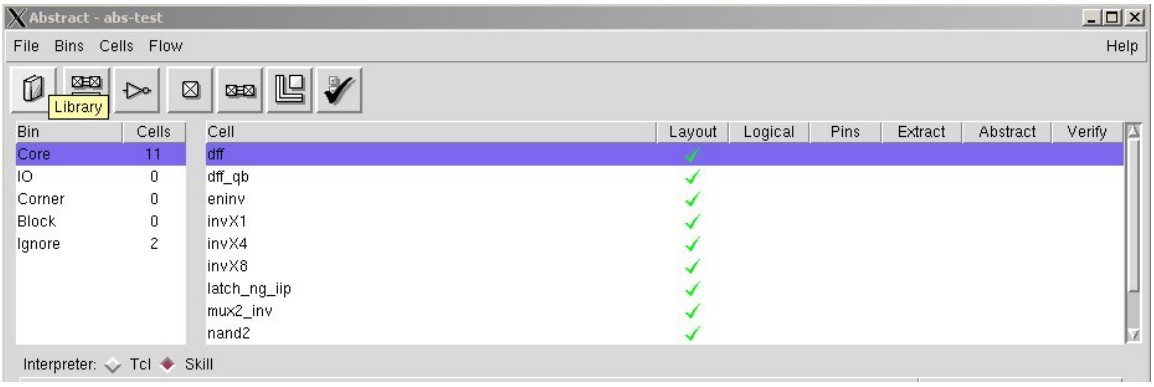

Figure 9.3: Cells in **abstract** after the library has been read

In particular, you should add a string that defines which text layer on top of which routing layer should be extracted as a signal pin. For example:

((text drawing)(metal1 drawing)(metal2 drawing))

This text says that any **(text drawing)** layer that overlaps either **metal1** or **metal2** should be extracted as a pin.

2. You need to add any names you used for clock pins in your design.
3. You need to add any names you used for output pins in your design.
4. The regular expressions for power and ground should default to something that recognizes the **vdd!** and **gnd!** labels that we use. If you've used something else, you need to modify these settings.

In the **Boundary** tab of this dialog box, you need to make sure that the geometry layers that you want **abstract** to use to generate the bounding box are set correctly, and that the **Adjust Boundary By** fields are set to **-1.2**. Our standard cell format requires that the bounding box is smaller than the geometry by 1.2μ on each side. This way the cells will overlap slightly when they are abutted according to the bounding box. This tab is shown in Figure 9.5.

When both of these modifications are made, you can click **Run** to run the **Pins** step of the process.

When I run the **Pins** step I always get a lot of warnings. In particular you'll probably see **ABS-515** warnings because the generated bounding box doesn't enclose all the layout geometry. Of course, we told **abstract** to do it that way in the **Boundary** tab of the **Pins** dialog! You'll also probably get **ABS-502** warnings that no terminals were created in your cell. If you used

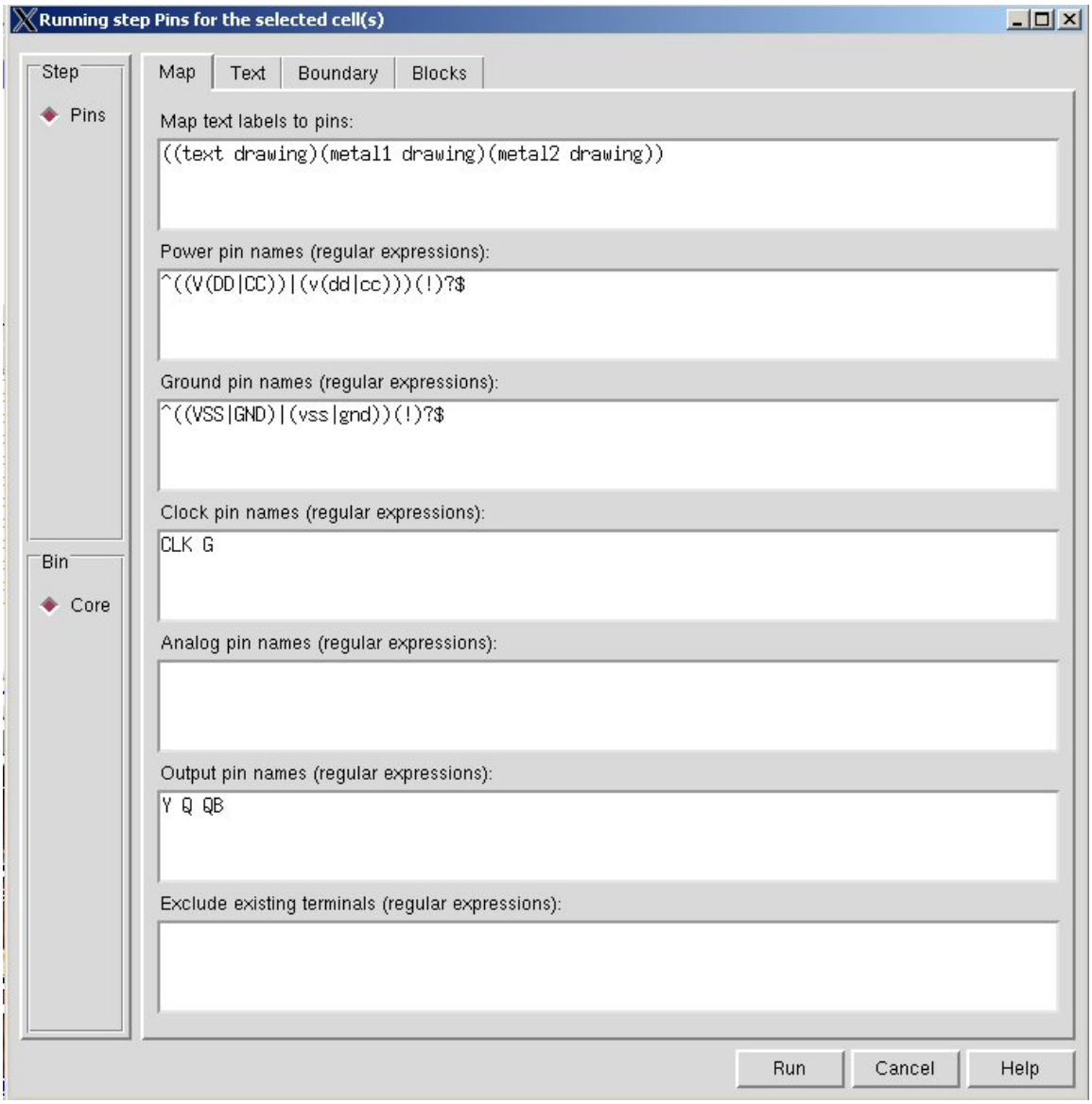

Figure 9.4: The **Map** tab in the **Pins** step dialog box

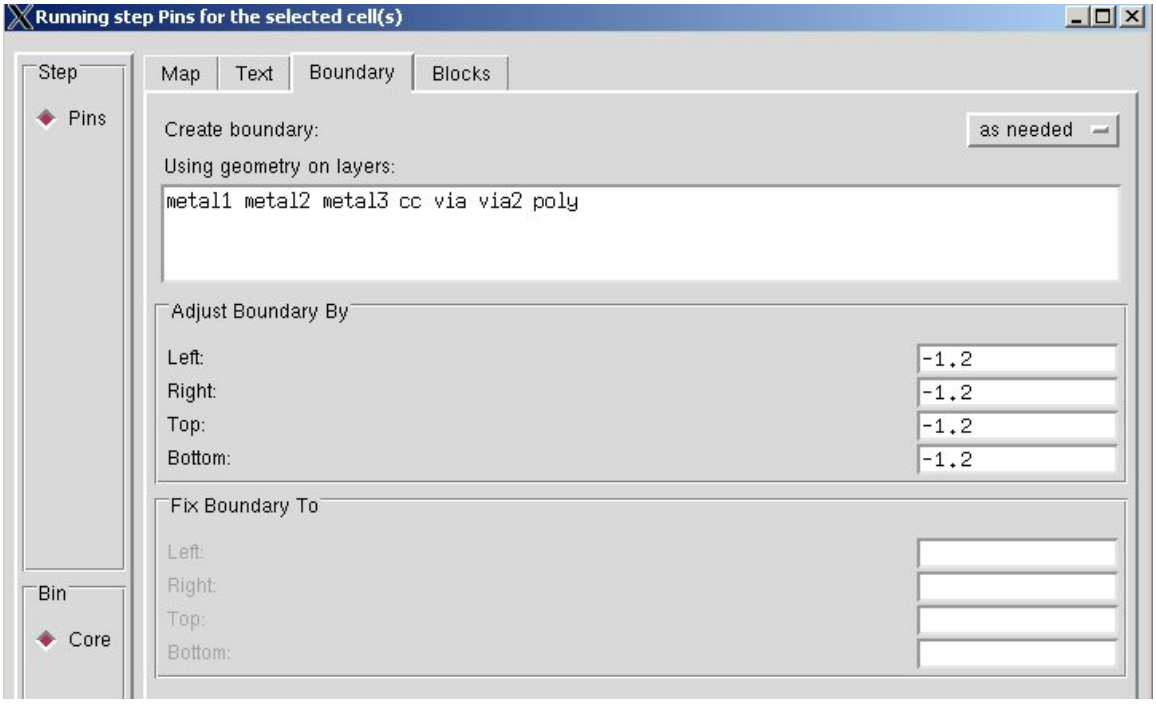

Figure 9.5: The **Boundary** tab in the **Pins** step dialog box

shape pins correctly then you can ignore this. You should see a previous line that says that the terminals already exist. All this warning is telling you is that the terminals already exist as **shape pins** and nothing had to be extracted through the text label technique. I also sometimes see warnings about **rcbScan** failures. These can also be ignored. They are minor issues with the database that don't effect behavior.

However, the result of all these warnings is that you're likely to see the pins step have all orange exclamation points!

9.1.3 The Extract step

The next step is to **Extract** the connection and obstruction information about your cells. The extract step widget looks like two contacts connected by a little wire. The dialog box is shown in Figure 9.6, and you shouldn't have to modify it from the defaults. The result of running this step should be green check marks in all rows of the **Extract** column.

9.1.4 The Abstract step

Finally you can generate the actual **abstract** views of your cells with the **Abstract** step. In the dialog box you can leave most things as defaults. In the **Site** tab you should change the **Site** to be **core** (see Figure 9.7. You can check the other tabs if you like. The **Blockage** tab should have all the routing layers specified as **Detailed Blockages**. The **Grids** tab will have **metal1** and **metal2** routing grid information that was derived from the technology file. In particular the **metal1** pitch will be 3μ because it is helpful if it matches the wide pitch of the other horizontal routing layer **metal3**. The **metal2** vertical pitch is smaller at 2.4μ.

If each of these steps has been run correctly you should see something similar to Figure 9.8. We skip the **Verify** step because it involves setting up test scripts in the place and route program which we haven't done. Instead we now generate the output file.

9.1.5 LEF File Generation

The result of this process is that the **abstract** views of the cells can now be output in a **Library Exchange Format** or **.lef** file. This is an ASCII-formated file that contains a technology header and then a description of the abstract views in a format that the place and route tools can understand. You can create this **.lef** file using the **File** → **Export** → **LEF** menu. In the dialog box (Figure 9.9) you should export only the **Geometry LEF Data**

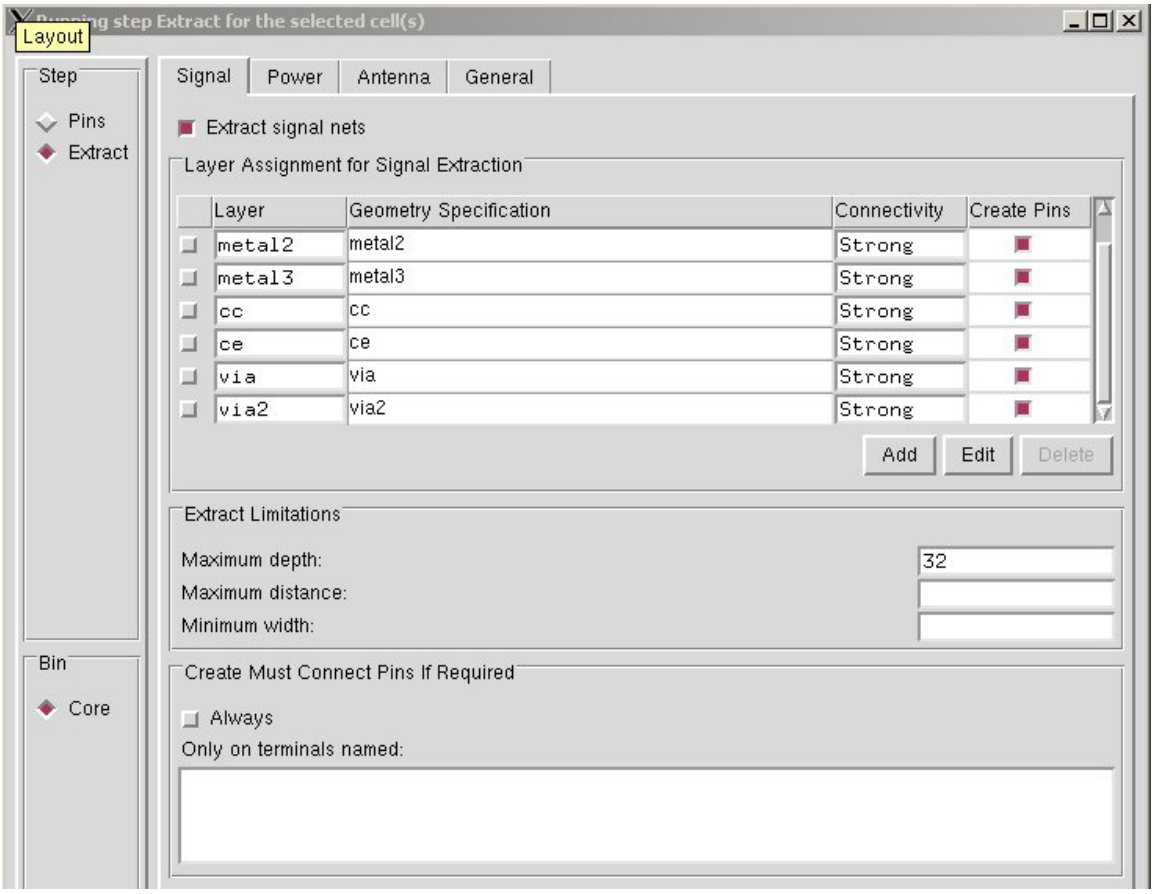

Figure 9.6: **Extract** step dialog box

Figure 9.7: **Site** tab in the **Abstract** step dialog box

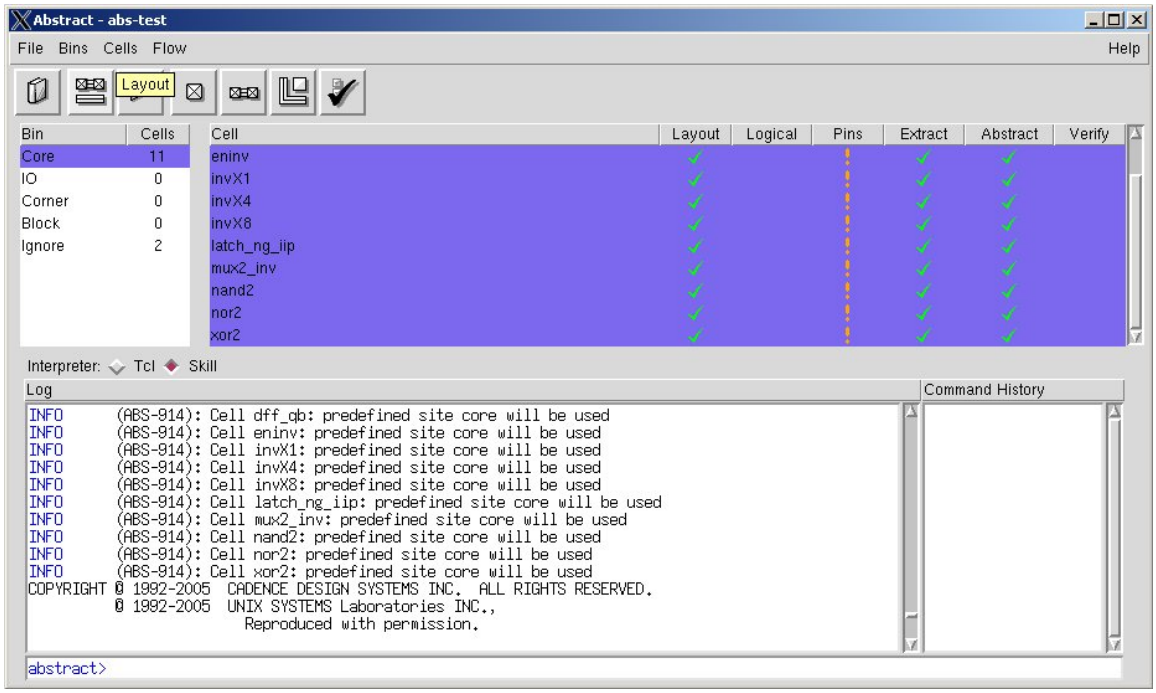

Figure 9.8: **Abstract** window will the **Layout, Pins, Extract** and **Abstract** steps completed

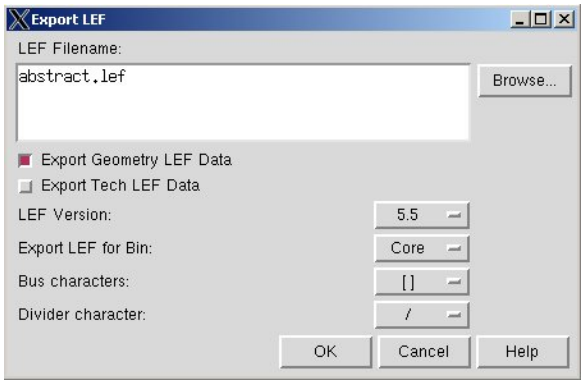

Figure 9.9: Dialog box for exporting **LEF** information

(We'll add our own customized tech data later), make sure that the **Version** is **5.5**, and the the **Bin** to be exported is **Core**. You can name the exported file anything you wish.

Once you have successfully exported the **.lef** file you can exit the **Abstract** program.

9.1.6 Modifying the LEF file

When you export the **Geometry LEF Data** you are exporting only the information about each of your cells. You want to collect these macro descriptions into one big **LEF** file that describes your entire library, but with only one copy of the **tech** header information. Also, we've added some extra information into the **tech** header beyond what gets generated by **Abstract**. You can get the **tech** header from **/uusoc/facility/cad_common/local/Cadence/lef_files/TechHeader.lef**. This file is also listed in Appendix E.

When you're ready to assembly your complete **LEF** file for your full library you should start with one copy of the **TechHeader.lef** file, and then insert only the **MACRO** definitions from your use of the **Abstract** program. That is, do not repeat the **VERSION**, **SITE** and other info in your final **LEF** file.