CADENCE DESIGN SYSTEM TUTORIAL

By
Rashmi Y. Dinakar
Bryan S. Goda
John Mayega

e-mail: kraftr2@rpi.edu
(for questions and comments)

ECSE 4220: VLSI Design
Rensselaer Polytechnic Institute
4 November 2003

Chapter 1: Introduction to Cadence

1.1 Introduction

Cadence Design Systems provides tools for different design styles. In this tutorial you will learn to use three Cadence products: Composer Symbol, Composer Schematic and the Virtuoso Layout Editor. This tutorial will help you to get started with Cadence and successfully create symbol, schematic and layout views of an inverter. You will also learn how to simulate your design using Hspice. The final check will be seeing if your layout matches your schematic.

Figure 1.1 shows the normal design sequence from design specifications to final layout simulation. This tutorial will take you through all the steps (except the last). In addition, there are chapters on Verilog, VHDL, bipolar current mode logic (CML), standard cells, and auto placement and routing.

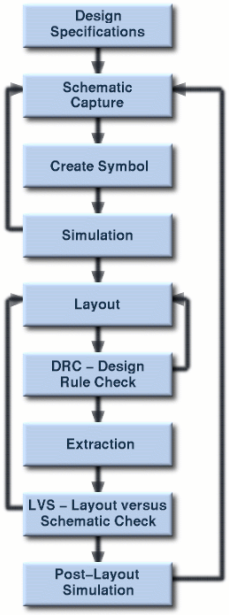


Figure 1.1: Design Process Flow Diagram

1.2 Getting Started

On the login screen enter your USERID and PASSWORD. To set up your ECSE ECL account for Cadence, Verilog, VHDL, Hspice, and AvanWaves, first logon to an ECL Sun using your account name. Type `ls -al` and verify that the following seven files are in your home directory. Missing or corrupted files may be replaced by typing:

```
cp ~cadtest/CDS/IC4.45-AMS2.4/.cdsinit .cdsinit
cp ~cadtest/CDS/IC4.45-AMS2.4/.cdsenv .cdsenv
cp ~cadtest/CDS/IC4.45-AMS2.4/cds.lib cds.lib
cp ~cadtest/CDS/IC4.45-AMS2.4/display.drf display.drf
cp ~cadtest/models.inc models.inc
cp ~cadtest/skew.file skew.file
cp ~cadtest/verilog verilog
```

A preconfigured .cshrc file should also be in your directory. If it is missing or if you have customized it for other applications, you may have to rename it and replace it by typing:

```
mv .cshrc .cshrc-old
```

```
cp ~cadtest/.cshrc .cshrc
```

In general you should keep a backup of any files you change in the same directory you are working in by copying the original file (`cp filename filename-old`) each time before making changes to the file.

Close your terminal window by typing:

```
exit
```

and then open a new one by holding the right mouse button down on a blank part of the screen, dragging down to **'Applications'** and then dragging down on the pop-up menu to **'Terminal'**.

If you don't want to replace or change your existing `.cshrc` file, before running Cadence or Hspice you will need to type:

```
source /cad/rc_scripts/rc.cadence_ic+icc
source /cad/rc_scripts/rc.hspice
```

1.3 Starting Cadence

Follow these steps to load the Cadence software.

- Move the pointer into a xterm/terminal/console window. The window borders change colors, which means that the window is active and ready to accept your commands.
- To load the Cadence software, type: **`icfb &`.** The ampersand (&) puts the command in the background, so you can continue using the window for other commands. The software is loaded when a new window opens up on the screen and the message "**END OF USER CUSTOMIZATION**" appears on the screen. This new window is called the Command Interpreter Window or CIW and is described in the following section.

1.4 About the CIW

The Command Interpreter Window is the first window that opens up when you run Cadence. Figure 1.2 shows the CIW and a brief description of its parts.

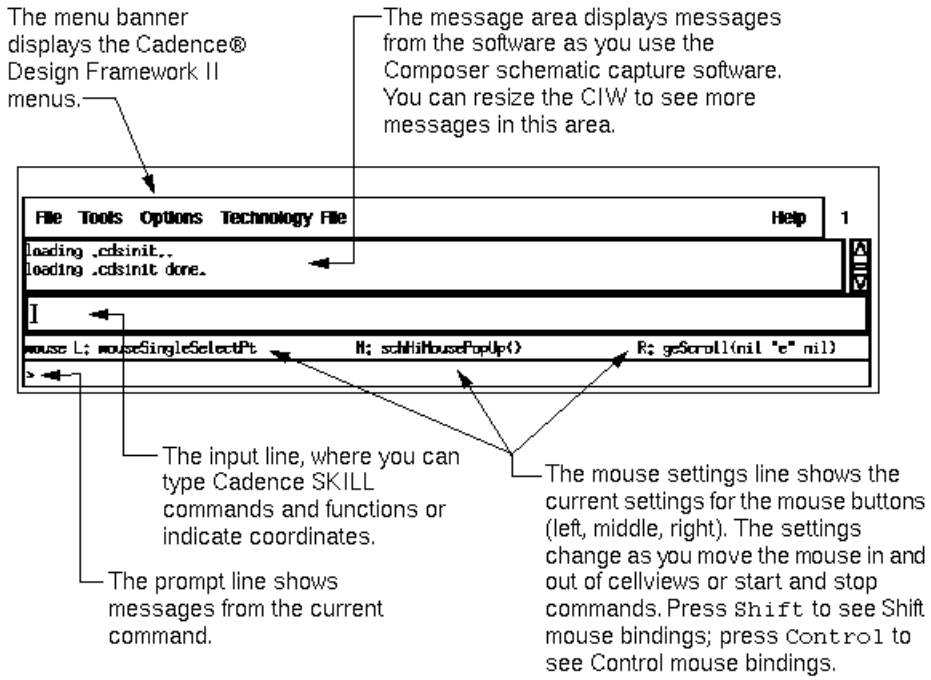


Figure 1.2: The Command Interpreter Window

In this tutorial you will learn how to create the symbol, schematic, layout views of a design and, how to simulate the design. In Chapter 2 and 3 you will learn how to create the symbol and schematic views of design while in Chapter 4 you will learn how to simulate your design. Chapter 5 shows you how to layout your design Chapter 6 checks your schematic against layout to see if it matches your design.

1.5 Getting On-line Help

Help for Cadence is available online. Cadence has an extremely friendly user interface. Every window that opens up when you run Cadence has a **Help** button on the menu bar. Clicking on this button will open up the help manual at the page containing the information relevant to the window. For example, clicking the **Help** button on the *Copy* form will open up the manual at the page containing information relevant to the Copy command.

1.6 Invoking Commands

Commands can be invoked in Cadence in three ways:

1. Using the items on the menu bar of each window:
   This is the easiest way to invoke commands in Cadence. As mentioned earlier, Cadence has an extremely friendly user interface. Most of the Cadence menus are pull-down menus. You click a menu title to pull down the menu and see the commands listed on it. Some commands have forms that must be filled up to supply the additional information that the command needs. Some commands have option forms that you do not always need. You can use the User Preferences command to set whether or not option forms appear when commands start. To do this,
   i. In the CIW, click **Options**. The dots after a menu item mean a form appears after you click the item.
   ii. To close the Options menu without starting a command, click anywhere outside the menu. If you accidentally choose a menu item, click **Cancel** to cancel the command.
   iii. Click **User Preferences**. The *User Preferences* form appears. This form contains settings that control how Cadence behaves. The form has two main sections, Window Controls and Command Controls. The items on the form are self-explanatory. In the command controls section, observe the "Options Displayed When Commands Start" option. This is the option, which determines whether or not a form opens up when you invoke a particular command. If this has not been selected, select it. Also, it is useful to set the **'Undo Limit'** to its maximum value to 10. This has all ready been setup for you in the .cdsinit file startup file. If not, make the necessary changes.
   iv. If you have made changes that you would like to preserve, click **OK**. Otherwise click **Cancel** to avoid changing the settings in this form.

2. Using Bind Keys:
   Letters and symbols to the right of a command describe a "bindkey" or set of keyboard keystrokes that perform the same function. The caret (^) represents the Control key: press the control key at the same time that you press the letter. Bindkeys are case sensitive. Make sure you know when the command letter is capitalized. Appendix A contains the Quick Reference for bindkeys that are supplied with Cadence. Using bindkeys is the fastest way to work with Cadence but, it requires a degree of familiarity with Cadence design environment.

3. Typing the corresponding skill function at the prompt in the CIW:
   This is an advanced way of invoking commands in Cadence and requires familiarity with the Cadence Design System and with the skill functions. Hence it is recommended that you use either the items on the menu bar or, the bind keys. If a skill function must be used, it will be mentioned at the appropriate place.

1.7 Creating a New Library

Take a look at the existing libraries. Select **Tools → Library Manager**. You should see the following (among other libraries):

sigehp	IBM Design Library
analoglib	Cadence Library, has ideal sources for voltages (vdc, vpulse)
VLSI_CLASS	Class examples of inverter, NAND gate

In order to separate your files from those that already exist in the system, you must create a library of your own and place your files in that library. Library creation is a critical step. **You must do this exactly as stated under AMS utils, not under any other pull-down menu.** If this procedure is not followed, you will not be able to edit the circuit layouts. Everything in the diagram will be collapsed down to a single layer. To create a new library from the CIW, do the following:

- Choose **AMS utils → Library → Create** from the CIW.

The *New Library* form opens up. In the form

- In the **'Name'** field, specify the name of the library
- **'Technology File'** field: select **Attach to an existing techfile**
- Select **OK**

The *Attach Design Library to Technology File* form opens. In this form

- Select **sigehp** for the **'Technology Library'** field
- Select **OK**

An *Add AMS Library Properties* form opens. In this form

- Select **M5** for 5 layers of metal for **'LM level**' (this is important otherwise your layouts will be wrong)
- Select **OK**

1.8 Opening a New Cell View

To open a new cell view from the CIW, do the following:

1. Choose **File → New → Cellview…** from the CIW.
2. The *Create New File* form appears.
   - In the **'Library Name'** field, specify the library in which you want the new cell view to be stored. (Pick the library just created.)
   - In the **'Cell Name'** field, type a new cell name.
   - Set the **'Tool'** field to either **Composer-Symbol** / **Composer-Schematic** / **Virtuoso** depending on the view type. (For the first time through, pick **Composer-Schematic** in preparation for Chapter 2 of this tutorial. Chapter 3 will use **Composer-Symbol**.)
   - In the **'View Name'** field, a view name - **symbol**, **schematic**, or **layout** - depending on what you selected under **'Tool'**, will be given.
   - Click **OK**

1.9 Opening an Existing Cell View

To open an existing cell view from the CIW, do the following:

1. Choose **File → Open** from the CIW.
2. The *Open File* form appears.
   - In the **'Library Name'** field, specify the library that contains the existing cell view.
   - In the **'Cell Name'** field, specify a cell name by doing one of the following:
     - Type the cell name
     - Click on a cell name in the **'Cell Names'** list box
     - Click **Browse** to open the Library Browser.
   - In the **'View Name'** field, choose a view name.
   - Click **OK**

Figure 1.3 shows what a typical design window looks like.

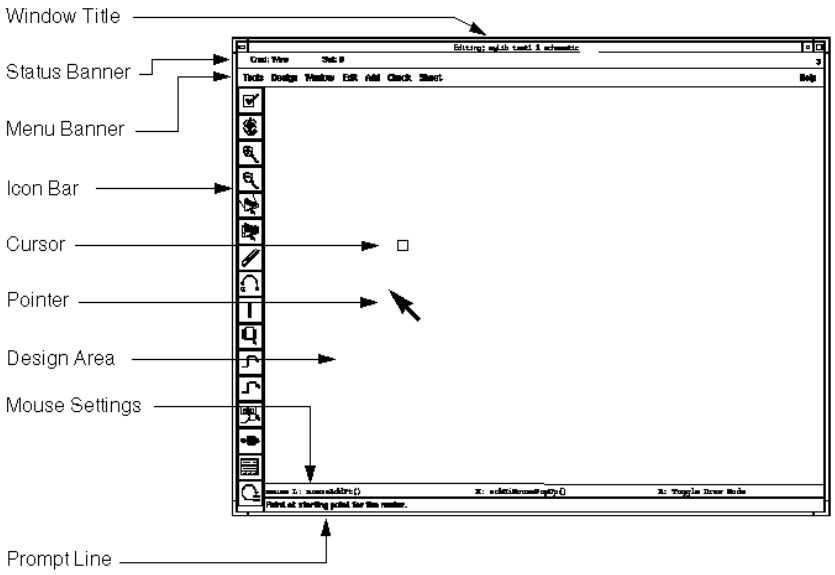


Figure 1.3: Design Window

1.10 Capturing Screen Images with XV or Snapshot

As you proceed through this tutorial as part of the VLSI Design class you are expected to keep a record of your results after finishing each section. A convenient way to do this is by capturing images off the screen after each point in the design. Suppose you have a nice plot you want to save. You can type: **`xv &`** to start the xv program. Right click on the box and select **Grab**. A box should appear. Put in 2 or 3 seconds for the delay, hit **AutoGrab**, and move to the display window you wish to capture. You can left click to get the window or draw a box with the middle mouse button to select a rectangle you wish to view. You should see a copy of the box appear if you grabbed it correctly.

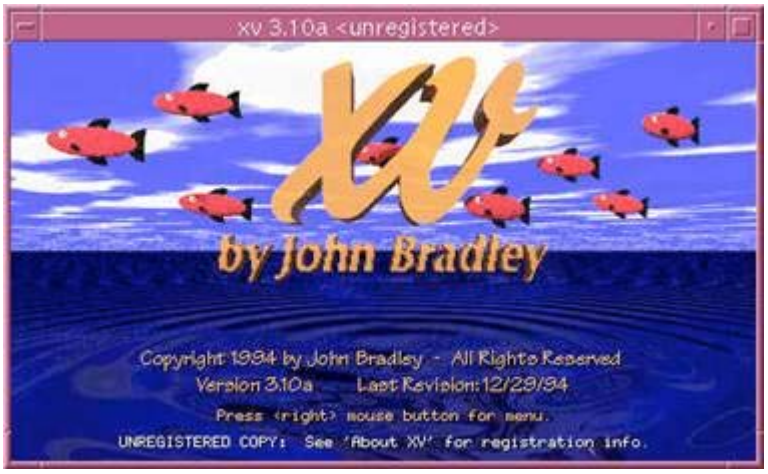
If you want to play with the colors, select 8 bit mode, then color editor. You can click on a color box and change it to meet your tastes. If you want to flip black to white, slide the dials (all 3) to 255. You can also play with the saturation to bring out the contrasts. It is strongly suggested that you remove any black backgrounds since they don't print well, waste a lot of ink, and make it difficult to see the important features in your drawings. Save the file as a full color .gif or .jpg and then later you can ftp it to your RCS account and port it into PowerPoint or Word.

For a step-by-step procedure to grab screen shots:

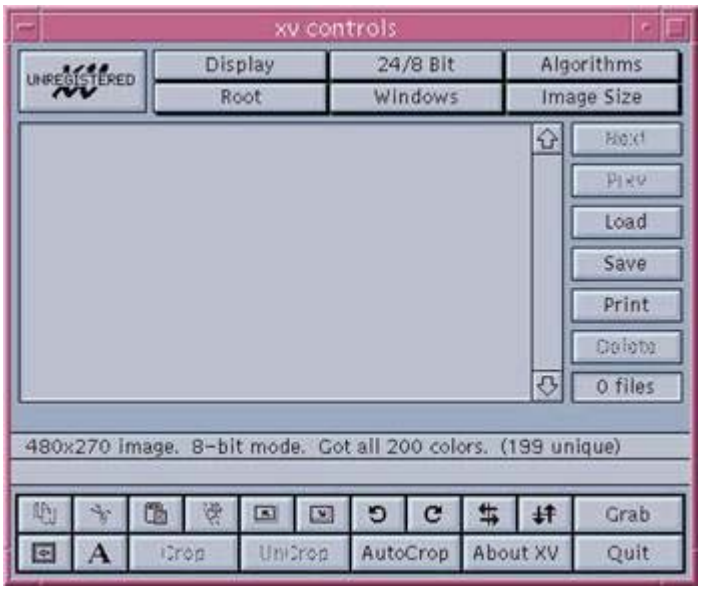
1. Create a directory for saving your screen capture pictures in the Unix window.
   ```
   mkdir pics
   ```

2. Start the screen capture software using command:
   ```
   xv &
   ```

   This window will open on the terminal:



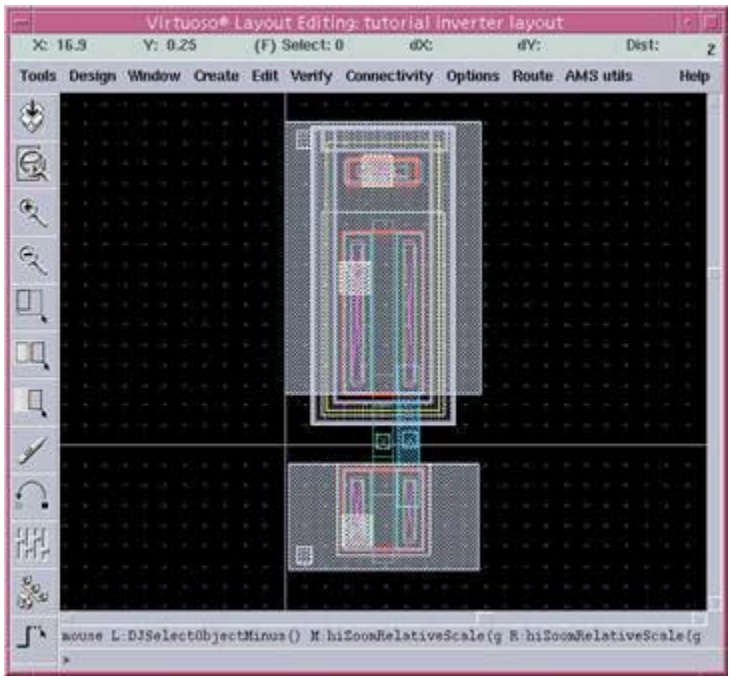
3. **Right clicking** on the XV window will show:



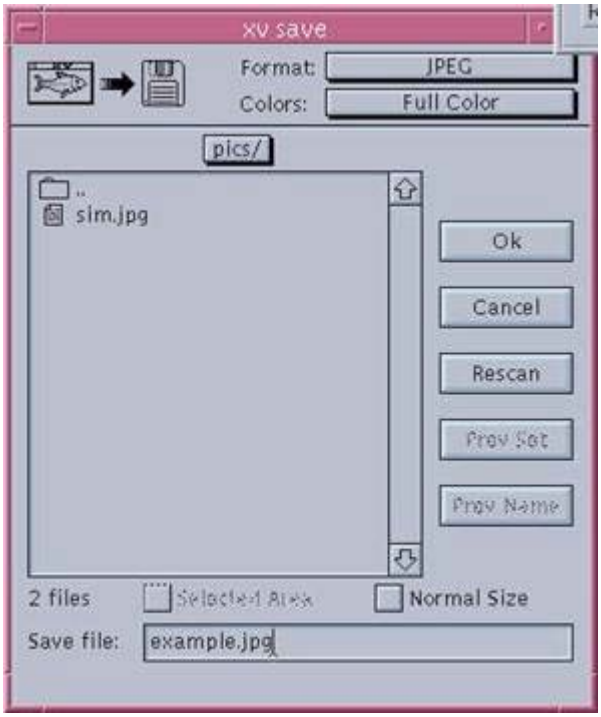
4. **Left click** on the **Grab** button and up pops the screen:

5. You can use **AutoGrab** or **Grab** to capture the window or picture you want. Here, the **AutoGrab** will be used. For this example, fill in 3 seconds for the delay.

6. The **AutoGrab** button is enabled by **Left clicking** on it. You must activate (move your mouse to that window's title bar and left click) the window you want to capture within 3 seconds. E.g. we choose the inverter layout window. It will be captured as shown below.



7. After you have captured the window go to the XV control window, shown in step 3, and click **Save** to save the picture:



Note: don't forget to choose the correct folder in which you want to put your picture. Use a JPEG, GIF, or TIFF file format. You should add a .jpg, .gif, or .tif suffix to the saved file name.

9. Repeat steps 4 through 8 to capture all windows in which you are interested.

10. Use the following command to transfer the pictures you captured to your RCS account so that you can print them out in the VCC or insert them into a Microsoft Word document.

```
scp -r pics username@rcs-sun.rpi.edu:
```

The ***username*** is your RCS account name. Don't leave off the ":" at the end of this command.

Another way to create an image file of the plots is as follows. **Right click** on a blank part of the screen and select **Programs → Snapshot**. Select **Hide Window During Capture** and then **Left click** on **Snap**. Move the cursor to the desired plotting window and **Left click**. Wait several seconds until the control panel reappears with an *Image Viewer* window of the captured plot screen. Select **File → Save As** in the *Image Viewer* window, pick a **'File Format'** (TIFF is probably best for PC and Macintosh files), enter a file name, and select **OK**. Note that this method will work for capturing an image of any window on the Sun workstation.

1.11 Further Reading

Searching the web will turn up other sources for Cadence tutorials. A comprehensive site is:
http://vlsi.wpi.edu/courses/ee390x/
Another is (but this cite seems to be down a lot): **http://www.cadence.ncsu.edu/tutorial.html**

1.12 Tutorial Caveats

Cadence is one of the most complicated packages many of you will ever use. It has many options and many different way of accomplishing the same objective. Because of this flexibility, this tutorial strives to be detailed in its step-by-step approach to guiding the user through the menus and tasks. A small change from

the procedure may unknowingly lead the user down a completely different path, but this fact is not immediately obvious because both paths will have similar menus and form windows. Using the exact names of menus and forms in the tutorial that appear in the program helps keep the user on track, but makes maintaining the tutorial very difficult. Cadence releases a new version of the package every year and requires universities to upgrade each time. In the process, each new release has minor changes to commands, menu items, form names, and the position of options or graphics in windows. The exact names described in the tutorial may not correspond to those used in the current release. We will strive to maintain high accuracy in this manual, but the user must be ready to decide whether a discrepancy between the tutorial and the on-screen display are due to a minor change in the program or a misstep in following a procedure – a very difficult decision for a beginning user indeed.

Chapter 2: Schematics

2.1 Creating a Schematic

A schematic is the graphical representation of the logic circuit design either by discrete devices (transistors, resistors, etc.) or by logic elements (AND gates, OR gates, etc.). The connectivity information is obtained from the placement of pins and wires. To create a schematic design, you use a schematic editor. The schematic editor window looks like the design window shown in figure 1.3. Follow the procedure in section 1.8 to create a schematic view if one does not already exist.

2.1.1 Adding objects to a schematic

Use the Schematic editor to add the following objects

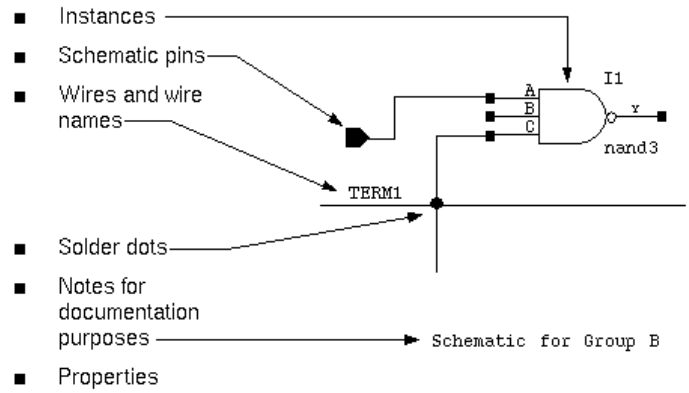


Figure 2.1: Objects in the Schematic Editor

The symbol editor is usually used to draw the symbols (simple electronic devices or logic elements) that are placed in the schematic window. The schematic editor draws the circuits connecting the symbols together. In this section you will draw the schematic of an inverter. A CMOS inverter consists of a nmos transistor and a pmos transistor. The symbols for the nmos and pmos transistors have already been provided with the IBM SiGe design library. You must use these symbols to create your inverter circuit. To create the schematic:

1. In the library manager window pick **File → New → Cellview**. In the *Create New File* form:
   - Select your working directory
   - Fill in the name field
   - In the **'Tool'** field select **Composer-Schematic**
   - Click **OK**
2. The design window opens.
   - On the menu bar click on **Add → Instance**. The *Add Instance* form appears.
   - Click on **Browse**. The *Library Browser - Add Instance* window opens up. In this window, select **'Library'** = **sigehp**, **'Category' = FET** (if **Show Categories** is on), **'Cell'** = **nfet**, **'View'** = **symbol**
   - Move the cursor onto the *Composer-Schematic Editing* (or *Virtuoso® Schematic Editing*) window. You will see the symbol attached to the arrow pointer. A list of characteristics will be added to the *Add Instance* form. Edit, if desired, the **Width**, **Length**, and **Substrate node** (use sub!). **Left click** to place on the drawing sheet.
   - Repeat the process to get a pfet. For a pfet the **Nwell node** should be the highest voltage (normally Vcc or Vdd)
   - Place the input and output pins connections for the inverter with **Add → Pin** (or use the shortcut key **p** to add a pin). An *Add Pin* form opens. Set '**Bus Expansion**' to **off**, '**Placement**' to **single**, '**Direction**' to match the schematic signal direction (**input**, **output**, or **inputOutput**), and '**Usage**' to **schematic**. The symbol will move with the arrow pointer and a **left click** will add it to the drawing

- To add wires **Add → Wire (narrow)**. The following section describes wire to pin connections and pin to pin connections in detail.
- To save a design choose **Design → Save** or **Design → Save As…** and click **OK**.
- To close the editor window do the following: **Window → Close**. If you have not saved your design, the *Save Changes* form appears and prompts you to **Save the changes**, **Discard** them, or **Cancel** the Close operation.
- Use the wire drawing tool to connect up your design. Use the wire name button (13th down on left side) to name Vcc! and Vee!. The ! means a global net that can be seen at all levels. This is especially useful for wiring power and grounds. Make your schematic match the example in the library: **VLSI_CLASS, inverter, schematic**. You can use the right mouse button as you are placing an instance to rotate the instance 90 degrees.

2.1.2 Making wire to pin connections

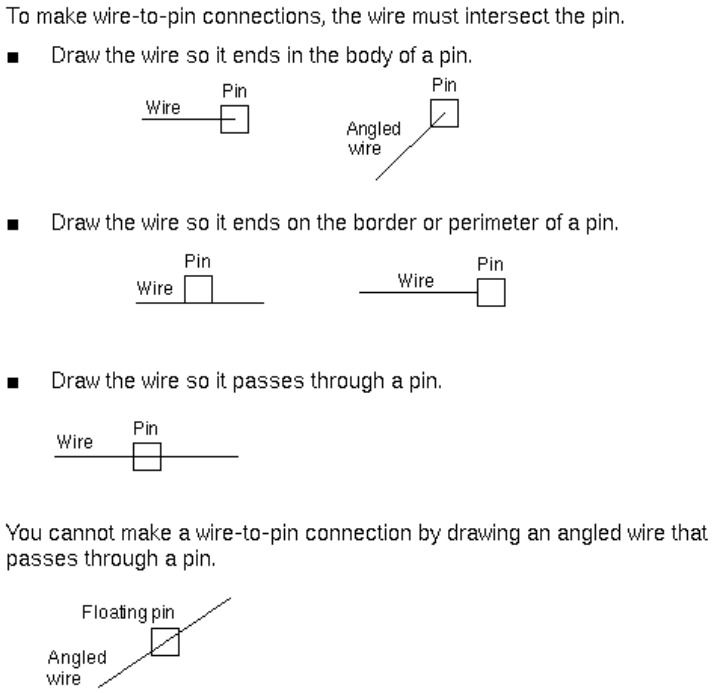
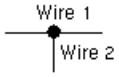


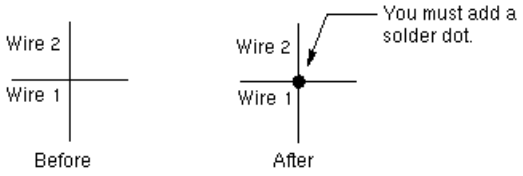
Figure 2.2: Making wire to pin connections

2.1.3 Making wire to wire connections

If you draw two intersecting manhattan wires in a T configuration, the editor automatically connects the wires.



If you draw two manhattan wires that cross, you must explicitly add a solder dot to make the connection at the crossover.



If you draw two manhattan wires in an L configuration, or if you draw an angled wire and a manhattan wire that intersect at their end points, the wires are connected, but the editor does not add a solder dot.

2.2 Editing Commands

2.2.1 Stretching

Use the stretch command to reposition the placement of objects in your design without losing connectivity between wires and pins. Direct manipulation is the easiest and fastest way to stretch objects or shapes. You use the mouse rather than commands to edit. To directly manipulate an object, do the following:

1. Click and hold the left mouse button over an object
2. Drag and release the mouse button to complete the stretch

The stretch command works differently with the schematic editor and the symbol editor.

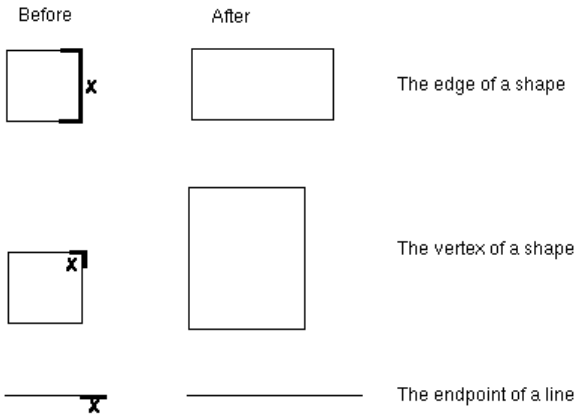
Stretching with the schematic editor:

1. Choose **Edit→ Stretch** (or shortcut key **s**)
2. Select the object you want to stretch. The point you used to select the object becomes the reference point.
3. Point at a reference point
4. Move the mouse, and then click a destination for the stretch
5. If you need to change the stretch options, use the *Stretch* form. This form must appear automatically when you choose **Edit → stretch**. If it does not, hit the **F3** function key.

Stretching with the symbol editor:

To stretch an edge or a vertex of a shape or the endpoint of a line in the symbol editor, do the following,

1. Choose **Edit→ Stretch**
2. Click once on the edge or vertex or line that you want to stretch
3. Move the pointer to stretch the part you selected and point to a destination for stretch. If you select objects other than lines, rectangles, polygons, circles, ellipses or arcs, they move instead of stretch. Partially selected shapes stretch instead of move. To partially select an object, position the cursor on the design window and hit the **p** key. The design window banner reflects the change to partial mode. The following diagram shows how to stretch partially objects in the symbol editor.



2.2.2 Copying

1. Select **Edit → Copy** (or shortcut key **c**). The following prompt appears in the CIW **"Point at the object to copy"**.
2. If the *Copy* form does not appear, place the pointer in the CIW, and press the **F3** function key. The *Copy* form appears.
3. Fill in the *Copy* form
4. Click on the object you want to copy. The object appears highlighted in yellow.
5. Position the cursor where you want to place the object and click.

2.2.3 Moving

1. Select **Edit → Move**. (or **m** shortcut) The following prompt appears in the CIW window **"Point at the reference point for move"**.
2. Click on the object you want to move. An outline of the object follows the pointer when you move it. The prompt in the CIW now changes to **"Point at the destination point for move"**.
3. Position the cursor where you want to place the object and click. The object moves to the new location and the outline disappears.

2.3 Other Useful Commands

2.3.1 Selecting objects to edit

You can select objects to edit before or after starting an edit command. If you select an object before you start an editing command, the system prompts you for a starting point called the reference point for edit. When you select an object after you start a command, the command remains active until you cancel the command by hitting the **Esc** key.

2.3.2 Canceling a command

Press the **Esc**(ape) key, click **Cancel** on a form or select another command.

2.3.3 Undoing a command

To undo a command, Press the **u** key or, choose **Edit → Undo**

2.3.4 Redoing a command

To redo a command, Press the **Shift-u** key or, choose **Edit → Redo**

2.4 Working with Windows

2.4.1 Zooming

Zooming In Manually

To zoom in manually:

1. Choose **Window → Zoom → Zoom In.**
2. Click your cursor to mark one corner of your zoom area and drag your cursor to draw a box around the zoom area.
3. Click your cursor to mark the other corner of your zoom area

Zooming In Automatically

To zoom in automatically by 2 (or **z** shortcut, **Z** for 5x shortcut):

Choose **Window → Zoom → Zoom In By 2**

Zooming Out Automatically

To zoom out automatically by 2 (or **o** shortcut, **O** for 5x shortcut):

Choose **Window → Zoom → Zoom Out By 2**

Zoom To Fit Automatically

To zoom to fit automatically:

Choose **Window → Fit**

2.4.2 Panning

Panning lets you reposition your design within the editor window. To pan, you can do any one of the following:

1. Panning using your cursor and the Pan Command
   - Choose **Window → Pan**
   - Click on the point that you want to appear in the center of the editor window
2. Panning using the arrow keys: Press the arrow keys corresponding to the direction you want to pan
3. Panning using the Editor Window Scroll Bars

2.4.3 Redrawing a window

Choose **Window → Redraw** (or **R** shortcut)

Chapter 3: Symbols

3.1 Creating a Symbol

In this section you will learn how to:

1. Create symbols to represent the main blocks in your schematic and add the primary input and output pins to the symbol. Since this tutorial uses the inverter as an example, you will create a symbol for an inverter, and associate the symbol with the previously completed schematic with nmos and pmos transistors, that will be used in schematic representations of the inverter.
2. Save your work.

You create a symbol by assembling elements in your design such as:

- Lines, rectangles, polygons, circles, ellipses, and arcs
- Pins used to connect the symbol to the rest of a schematic
- Labels to identify the symbol and pins
- A selection box to surround a symbol that lets you select an instance of the symbol in the editor
- Notes that consist of text and shapes to annotate the symbol

Figure 3.1 shows an inverter symbol and the elements that were used to create it.

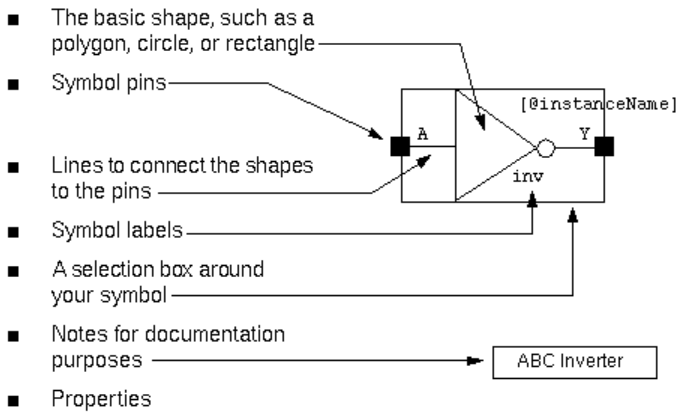


Figure 3.1: Symbol and its Elements

To draw lines for a symbol using the symbol editor, choose **Add → Shape → Line** from the menu bar. To draw rectangles for a symbol using the symbol editor, choose **Add → Shape → Rectangle** from the menu bar. Similarly you can also draw polygons, circles, ellipses, and arcs. To save your work click on **Design → Save** on the menu bar. To close the window click on **Window → Close**.

3.2 Creating a Generic Symbol for a Schematic

A simple procedure exists for creating a generic rectangular symbol for a circuit with inputs on the left, outputs on the right, and special pins on the top and bottom. In the schematic editor window, select **Design → Create Cellview → From Cellview…**. In the opened *Cellview from Cellview* form, keep all the defaults and select **OK**. The *Symbol Generation Options* form will open next. Again, the default values will work and you may select **OK**.

3.3 Creating a Symbol for an Inverter

In this section you will create a custom symbol for an inverter. The symbol is the first step in the design process. It represents the schematic, allows a hierarchy of design, and shows the inputs and outputs of each level. Symbols may be created that look like traditional logic elements, differentiating AND and OR gates, etc. To have a symbol associated with a specific schematic, it must have the same name, be stored in the same library, and have identically labeled I/O pins as the schematic.

Under the **Library Manager** select **File → New → Cellview** and then set **'Tool'** to **Composer-Symbol** in the *Create New File* window. A blank symbol editing window should appear complete with a grid of dots. This part of Cadence works much like a drawing program. You may also follow the instructions in section 1.8 to open a new cell view in your working directory. Since you will be creating a symbol be sure **symbol** is in the '**View Name**' field after selecting **Composer-Symbol** for the '**Tool**' field. The design window opens. The window title has your current working directory, the cell name and the tool type specified on it.

To draw lines for the inverter symbol, choose **Add → Shape → Line** from the menu bar. To draw the bubble, choose **Add → Shape → Circle** from the menu bar. To insert input and output pins for the inverter, choose **Add → Pin** (or you may also use the shortcut key **p** to add a pin). An *Add Pin* form comes up. Figure 3.2 shows the form.

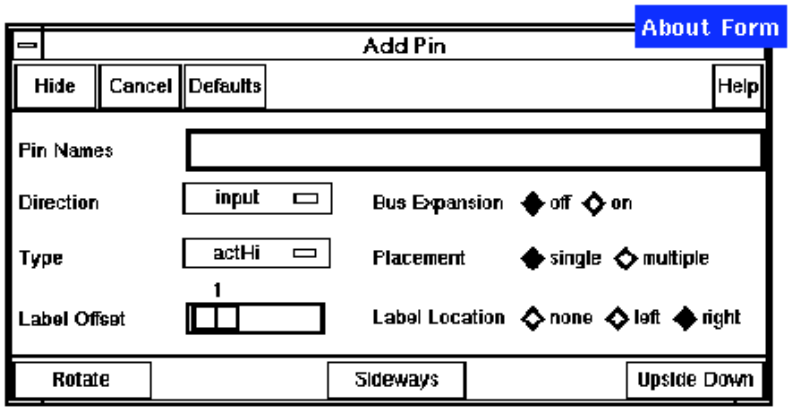


Figure 3.2: Add Pin Form

The '**Bus Expansion**' field is used with a bus and is set to **off** by default. The '**Placement**' field is used when multiple pin names are required. For the inverter symbol, leave the '**Bus Expansion**' field **off** and the '**Placement**' as **single**. For pin '**Type**' use **square**. The '**Direction**' must match the schematic signal direction (**input**, **output**, **inputOutput**). Type in a name and choose a '**Label Location**'. Now when you move the mouse over the drawing, an image of the pin will track the arrow and **left clicking** will add it to the symbol.

To add labels, choose **Add → Label** or hit the **[@abc]** button (13^{th} down (or last) on the left side). The default instance name will be [@instanceName] in the *Add Symbol Label* window. Place this in the center of your design. Later in the schematic view the instance name will be given as I#, where # is a number. For this, select **Add → Label**, select **logical label**, for '**Label Choice**' and enter the name of the symbol for '**Label**'.

To save your work, choose **Design → Save** from the Design Window menu bar.
To close the window select **Window → Close** from the Design Window menu bar.

You completed the symbol for an inverter.

Chapter 4: Hspice

4.1 Testing Your Schematic

A good design technique is to test your design at the top level. Create a new schematic called inverter_test. There is an example in VLSI_CLASS. In order to be able to edit the library VLSI_CLASS, you must be sure its pathname is correct. Open the *Library Manager* window (Click on **Tools → Library Path Editor…** in the CIW). In the *Library Path Editor:* form, make sure the path of the library VLSI_CLASS is set to "`~/VLSI_CLASS`". Click on **File → Save** and then **File → Exit** to save changes if you needed to fix it.

Some of the steps in the next few paragraphs may or may not be necessary. Due to the constants updates of both the version of Cadence IC and IBM AMS design kit, it is impossible to keep this tutorial and built-in example circuits completely error free. This section steps through typical errors that occur when the design kit is changed and what must be modified in the examples to fix the errors. Look at the schematic of the power supply example (**Power Supply**, **schematic**) in the library **VLSI_CLASS**. On opening up the schematic, a window may pop up giving you a message that a component could not be found. Close the error message window. You might find in the schematic of a power source the substrate connection named "subconv" crossed out. Delete this component only (**Edit → Delete** and **single left click** on it). Add the instance "subc" from the library **sigehp**, category **SUB** (if **Show Categories** is on), and cell **subc** in place of the older connection. Also edit the names of the +/- terminals to **Vdd!** and **Vss!** if necessary.

Instance your design, **Power_Supply** and **clock4ns_2on_2off** (from **VLSI_CLASS** library). You will need the power supply to set the reference voltages and the clock will be your test input signal.

Create a new cellview under your library. Enter the *Library Manager:* window. Select your library by **left clicking** on it. Select **File → New → Cell View…**. The *Create New File* form opens up. Enter the name of your test circuit in the '**Cell Name**' field. Ensure that the tool selected is **Composer-Schematic** and the library name is correct. Click on **OK**. The schematic editor window will open up.

Now you must add the components. Pick **Add → Instance…** and choose the three symbols; **Power_Supply** and **clock4ns_2on_2off** from **VLSI_CLASS** and the inverter from your library. Connect the **Clock** output of the pulse generator to the input of the inverter. Choose a capacitor from library **sigehp**, [**CAP**,] **mim**, **symbol**. Edit its properties (click on the capacitor, then **Edit → Properties → Objects…** or select the 9th icon down on left side) to change its '**Capacitance**' to **114f F** and the '**Backplate node**' value to **Vss!**.

Wire up your design like the example. You can look inside a symbol (to see the schematic) by **left clicking** on the symbol and typing **e** (for enter). To return, pull down **Design → Hierarchy → Return to top**. Take a look at the power supply and clock. While looking inside, select an item by drawing a box around it and selecting **Property** (9th icon down on left side or middle clicking). Here you can change voltages, rise times, pulse lengths etc. Notice how the power supply is constructed with the ground connection, DC sources, substrate contact, and pins. The 0 V DC sources are in place so you can see/measure currents in simulations. Save your design and check for errors.

4.2 Creating a Netlist

The following commands are to be performed from the menus in the schematic editing window. It is possible to perform the same operations from other windows since **Tools** also appears in the *CIW*, but the menu choices will be slightly different in some cases.

If your test schematic is error free, select **Tools → Analog Environment**. A new window should appear. Select **Setup → Simulator/Directory/Host**. In the new window select **hspiceS** for the **Simulator** option. It

takes about 30 seconds. Then select **Simulation → Netlist → Create final**. After 45 seconds a new window will appear with the netlist. Select **File → Save…** (or **Save As…**) and enter: **`~/inverter.sp`**. The .sp indicates a spice file and it will be saved in the top directory of your account.

4.3 Running Hspice

It is a good idea to put all of your simulations in a separate directory. Return to a Unix window, create a directory called sim (type: **`mkdir sim`**) and move your Hspice .sp file there. Copy the file ~cadtest/hspice_shell to your sim directory. Edit the .sp file using **`emacs`**, **`vi`**, or **`/usr/dt/bin/dtpad`**. Replace the last several lines with the corresponding lines from the hspice_shell file (you will have to change every line after the last `.ENDS` line). The lines represent the temperature, charge control model parameters, accuracy, and file links to the SiGe Hspice model. The most important line is the `.TRAN 1p 20n` line. TRAN means track the transient response starting at 1 ps to 20 ns. Change this line to meet your needs. Later you can vary this line to do voltage sweeps, step responses, temperature variations etc. Notice that the links to the models.inc and skew.file are in the directory level above the sim directory. Save your changes. Then type: **`hspice 'filename' &`**, where 'filename' is the name of the .sp file, and watch the fireworks. The data flying by is the model parameters for the SiGe HBT and CMOS transistors. Take Semiconductors Models and Devices I, II if you would like to understand more about how these models work. You should notice an **"Opening plot · · · .tr0"** file message, which is a good sign, since the .tr0 file is a trace file. After Hspice completes you should have a .tr0 file that is several MB in size. The name before the .tr0 suffix will be the same name that was before the .sp suffix. This file holds your data that AvanWaves can display for you.

4.4 Running AvanWaves (awaves)

Type: **`awaves &`** to run the viewer. When the plotting window opens, specify the data to be plotted by selecting in the window **Design → Open**. **Double click** on the folder containing your output file ('**sim**' in the class example) in the window that pops up. Select the *.tr0 file (where * is the name you gave the file) if it is not selected already. **Left click** on the **Filter** menu item. Make sure the box for '**Raw output**' is filled in. If not, move the cursor down to it and **Left click** on it. Select **OK** at the bottom of the window.

A *Results Browser* window will open. Under the label **'Curves:'** is a box containing plotting data names created in the simulation. **Double clicking** on a name plots it on the axis in the main window. Several waveforms may be plotted on the same axis simultaneously. Waveforms may be deleted by selecting the name on the '**Wave List**' and hitting the **<Del>** key or selecting **Panels → Delete Curves**.

Other possible plotting formats include **Panels → Add** to add another set of axes if you prefer to plot some values on another panel, **Panels → Delete Panel** to delete the selected panel (additional axis), and plotting currents if your simulation created the data. Functions also exist for adding labels to the plots, plotting mathematical expressions with the data, getting numerical values off the plot curves at specific points (and attaching a label to the plot at those points), changing the color of the plotted line, and rescaling the x and y axes.

When you are finished, quit AvanWaves with **Design → Exit**.

4.5 Capturing your Image with XV or Snapshot

Use the procedures described in section 1.10 to capture images of the waveforms plotted by AvanWaves for your report on this tutorial required by the VLSI Design class.

Chapter 5: Spectre

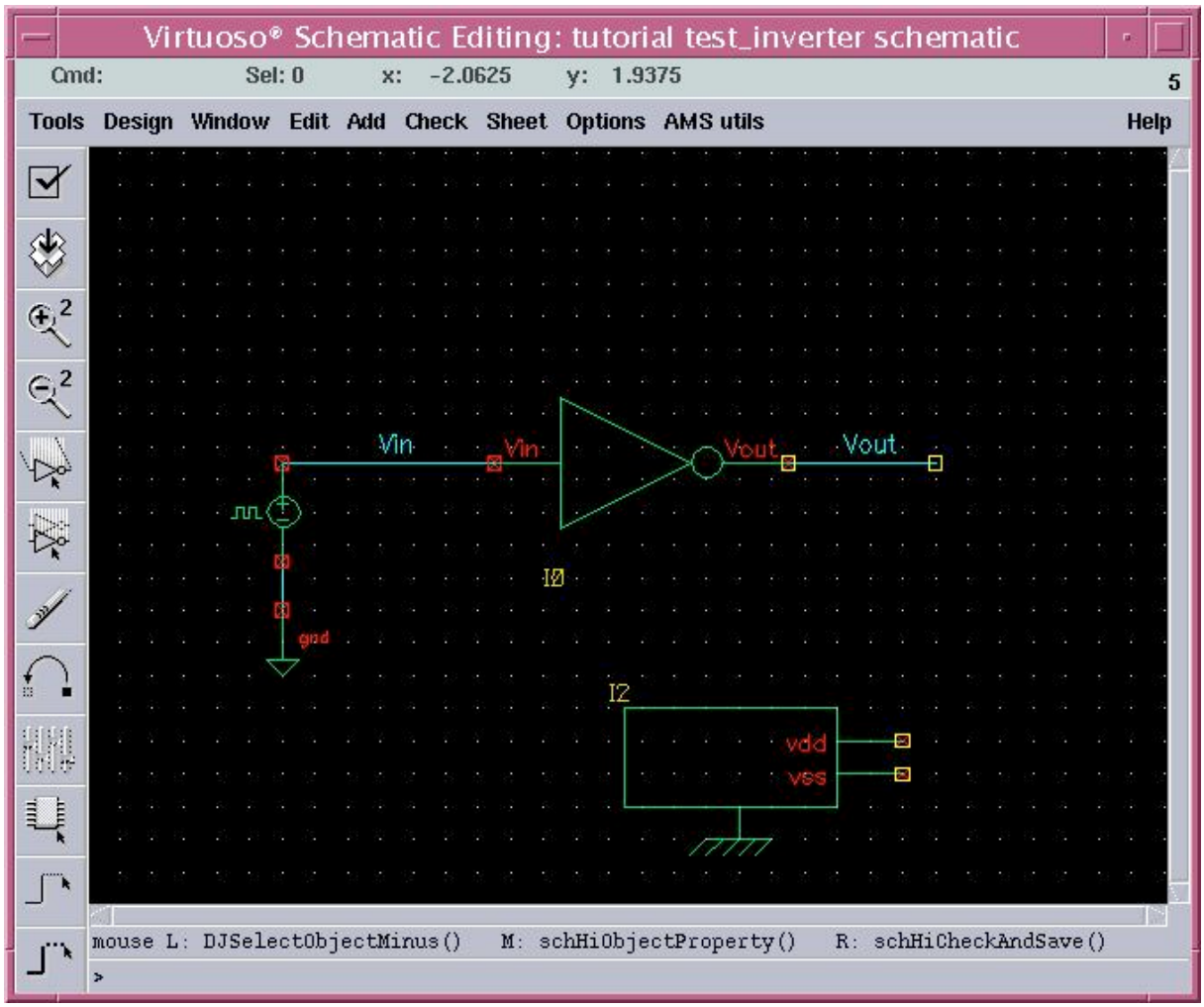
5.1 Spectre Introduction

An alternative simulation package to HSPICE is integrated with the Cadence tools. It is called Spectre. There are a few variations of this tool available. The chapter will discuss the **Spectre** and **SpectreS** tools.

5.2 Procedure

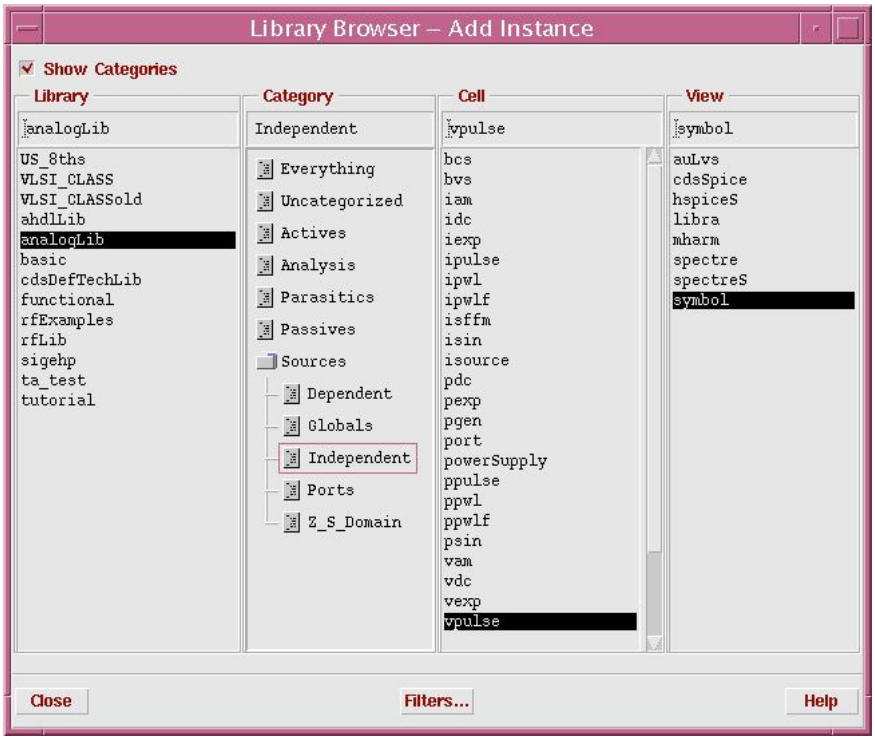
Usage is fairly easy for simple circuits.

1. To test whether your inverter works or not, we first create the test schematic as following.



There are three instances in this schematic.

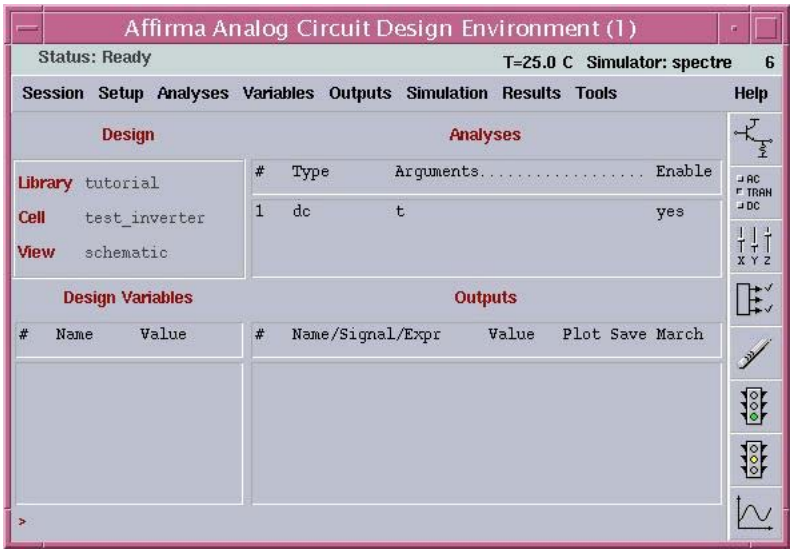
inverter: from the library "tutorial" which you just created.
power supply: from the library "ta_test".
vpuluse: from "analogLib" refer to below.



Specify your vpulse with DC voltage = 2.5 V; Voltage1 = 0 V; Voltage2 = 2.5 V; Pulse width = 2n s; Period = 4n s. Be careful to fill out these forms with the right format, otherwise your simulation won't run.

2. Make sure your schematic is error free; warnings like floating terminals are OK for this test.

From the schematic window you open **Tools → Analog Environment**, and the dialog appears. All the menu options explained here refer to that dialog unless explicitly stated otherwise.

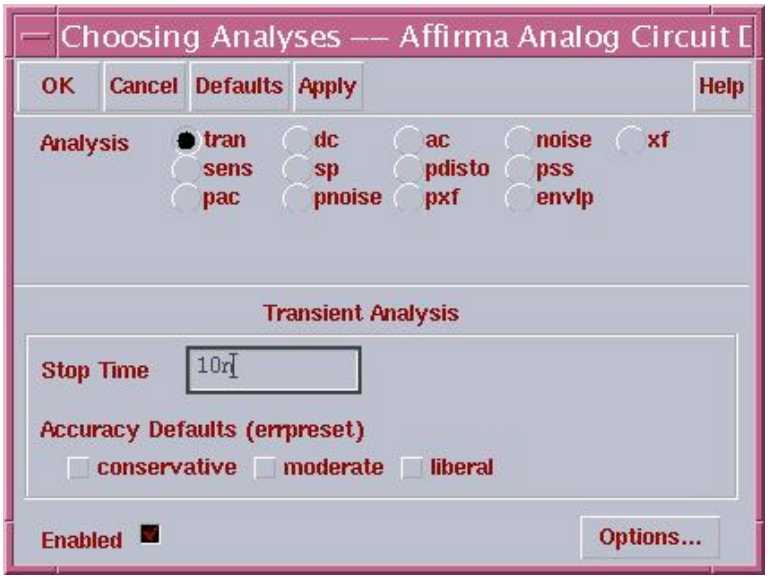


3. Choose **Setup → Simulator/Directory/Host**. This lets you choose "**Spectre**" or "**SpectreS**" as the simulator. The choice is not critical, but for reasons not important here SpectreS will be used instead of Spectre in this tutorial. The design kit environment should have been completely set up correctly to allow simulations, but if the default parameters need to be changed, then using **Setup → Environment** the init file, update file and include files from inside the IBM design kit can be specified. (They point to the models and have default temperatures, process parameters, etc.)

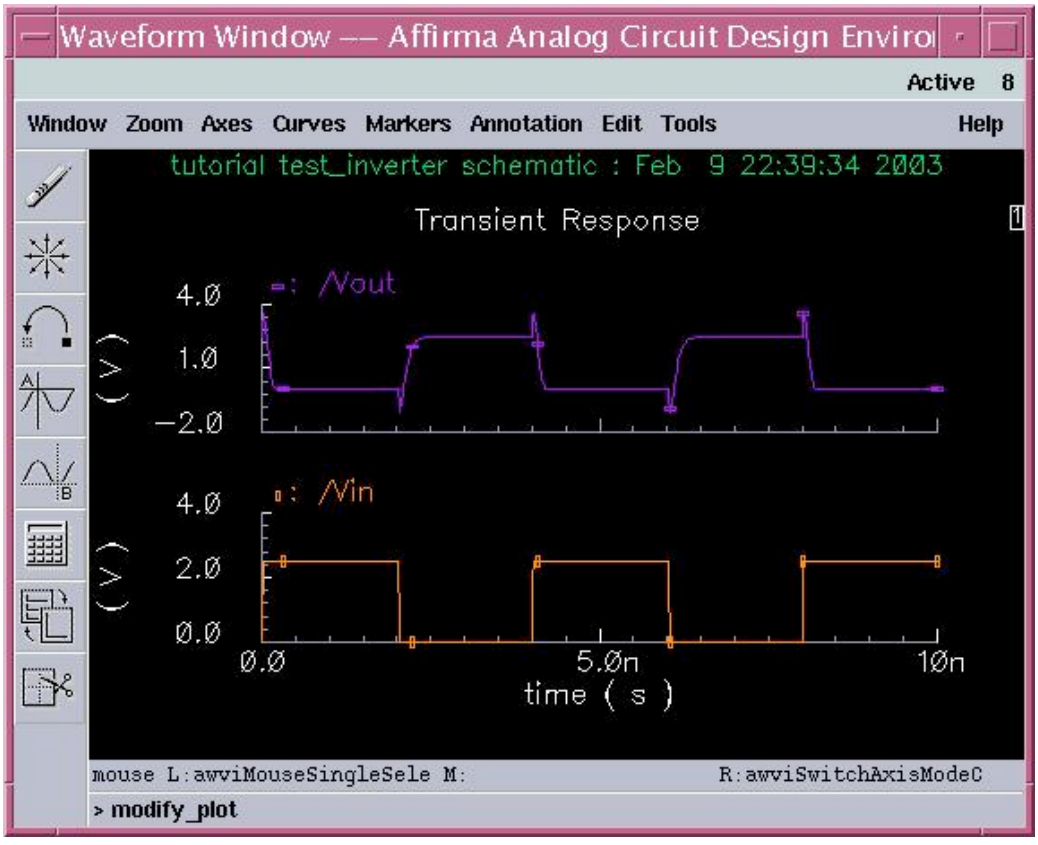
4. The menu selection **Analysis → Choose** lets you select a transient simulation with stop time, AC analysis, OP, etc. Multiple simulation types can be selected, and they will run in a reasonable order. If you want to be able to view anything other than voltages on the top level schematic you need to select them using **Outputs → To Be Saved**. You click on wires to record voltages, component nodes to record currents and hit **ESC** when done. You can descend the hierarchy as needed and add more signals. For really tiny circuits you can use the **Outputs → Save All** option.

Selecting items for a "marchfile" using **Outputs → To Be Marched**, stores those values into a text file, as opposed to saving them for plotting. The column ordering in the file is not apparent, and needs to manually analyzed to understand what is what. Only a limited number of signals can be saved in this way. (Some sort of maximum line length buffer limitation is in place.) The file is saved in the simulation directory.

In most component property dialogs, you can choose an arbitrary variable name to represent the value, and then use the **Variables → Edit** menu on the *Analog Artist* window to set it's actual value. This can be used to change a whole series of components simultaneously. Expressions like 2.7*TSIZE+5.0 in the components properties dialogs are also allowed. There are some places where variables **don't** work in the IBM design kits, for example in specifying pbdtres widths. These variables should be set using unit magnitudes like 'u', 'n', or 'M', but they don't take physical units like 'ohm', 'A', or 'F'.



5. Starting a simulation can be done using the "green light" button. When a simulation has started (gone past the OP point phase) you can begin to look at results using **Results → Direct Plot → Transient Signal**. You select multiple signals, hit **ESC**, and the plot window opens.



Optionally, you can do parametric simulations whereby the value of a variable you have chosen is stepped through some range of values. That is under **Tools → Parametric Analysis**. Pick a variable, a sweep type, range, etc and the simulation will run as many times as you have steps. When plotting the results of a parametric simulation, all the step curves for a selected wire are plotted at once.

Chapter 6: Layout

6.1 Layout Introduction

This is the most difficult part of the process. It would probably be a very good idea to look at the 2 examples before starting. (Examples are in VLSI_CLASS). Practice using the rules on the examples to get a idea of the dimensions. Play with the display controls. I normally set the display to view 10 levels. I think it starts out at 1 level, so the transistor doesn't look very interesting. Use the scroll bar to look at the various layers. These are described in Chapter 2 of the design manual. Here are the key ones you should know.

Types:

dg = drawing	(the most important and the type you use the most)
pn = pin	(used only for creating pins)
nt	only used in extracted view, don't draw in this layer.
br	for large vias. Must be 0.9 µ wide, 2.7 to 12 µ (microns) long

Minimum vias are done in dg .9 µ x .9 µ square. Vias are used to provide connections between metal layers.

Layer Names:

ca	metal one to polysilicon contact
mc	metal contact
rx	thin oxide layer (active transistor area)
nw	n-well (used for pfets)
dt	deep trench-used to isolate n-well around pfet
pc	polysilicon, used for gates and interconnecting gates (short runs, high R)
m1	metal 1, lowest layer metal
v1	via 1 connects m1 to m2
m2, m3 m4	other metal layers
v2, v3, v4	other interconnect vias
lm	last metal, use for power rails. Middle click on **lm dg** and then **Window → Redraw** to make this layer invisible.

There is a power rail example in VLSI_CLASS. You can open this window, draw a box around the rails, hit **c** to copy, and drag this to you layout window. A good idea is to use a standard set size for power rails, since you will be connecting cells together to make larger circuits. If you have a standard size then you can connect power and ground by just aligning last metal. You can use the stretch command to lengthen your rails. Make sure you turn lm back on if you want to be able to view the power rails.

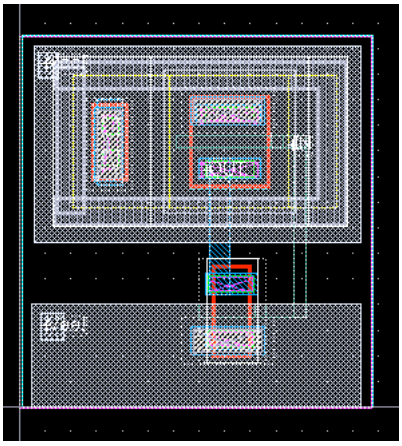


Figure 6.1: Inverter Layout

Figure 6.1 shows an example inverter layout. There are 2 transistors, a pfet (top) and an nfet (bottom) that comprise an inverter. You can instance these in the same manner as you instance objects in the schematic view.

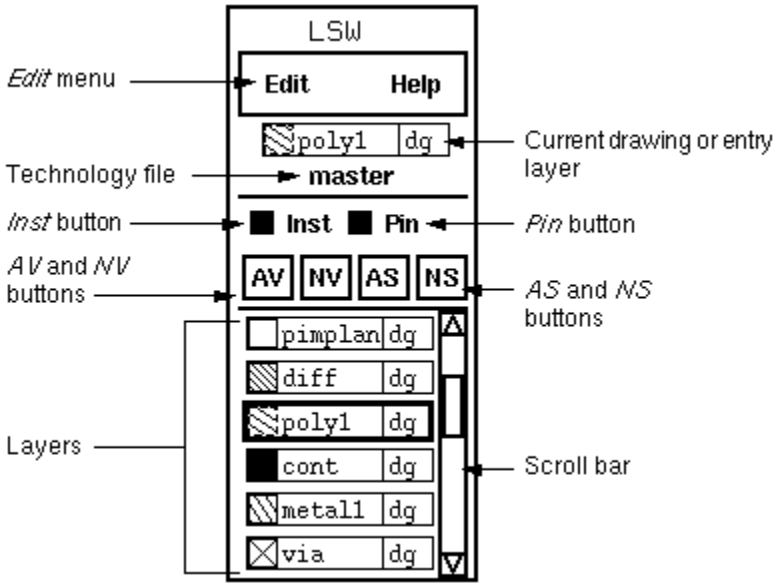


Figure 6.2: Layer Select Window (LSW) Note: layer names may be different

The LSW (Layer Select Window) allows you to display and select layers. The key buttons are AV (all visible), NV (none visible), AS (all select), and NS (none select). The current layer is selected by left clicking on it. If you want to turn off a layer, middle click on it and it should go gray. Then go to the layout window, hit **R** or **Window → Redraw** and the layer should become invisible. The best layer to turn off is lm (dg). Play around with the LSW features before continuing.

Open up the layout m1-lm in VLSI_CLASS. This is a connection between Metal 1 and last metal. You can make a copy of this layout in your library and then take it apart to see what is inside. This layout has metal1-via1-metal2-via2-metal3-via3-metal4-via4-lastmetal. The vias are created using the via# br layer, which is always 0.9 µ wide and at least 2.7 µ long (up to 12 µ). To make a min size via, use the dg layer, which is always 0.9 µ x 0.9 µ square. Notice there is a rectangle drawn around the layout called PR-Bound (dg) and Instance (dg) which you should place around each layout so you can instance them later on (as in our inverter example). Again, make sure you turn lm back on if you want to view it.

Make a copy of the inverter into your library. Turn off the lm (dg) layer and examine the deep trench (dt layer, gray in color), nwell (nw, yellow), the nwell contact (the funny looking rectangle off to the side), how the gates are connected with a pc (dg) run, and how power and ground are connected using the m1-lm instances. The pins in the layout are drawn by selecting the correct layer (must be the pn type on top of the layer you are drawing on) and then select **create → pin.** Fill in the pin name and the direction type must match the schematic. Draw a rectangle on top of the layer you want to draw on.

Choose **Window → Create Ruler**. The *Create Ruler* form appears. To measure the gate length of the transistor, click one edge of the poly1 layer. A ruler appears and grows as you move the cursor. The ruler measures objects in user units. The units are defined in a technology file. For this tutorial, the user units are set to microns (µ). To complete the ruler, click the opposite edge of the poly1 layer. To stop the command, click **Cancel** on the *Ruler* form or hit the **Esc** key. To remove the ruler you drew, choose **Window → Clear all Rulers.**

6.2 Creating a Layout

To create a new cellview, in the CIW choose **File → New → Cellview**. The *Create New File* form appears. In the form do the following:

Set the library name to **VLSI_CLASS**
Type **Inv** for cell name
Type **layout** for view name
Click **OK**.

Note: The tool field is automatically updated to Virtuoso.

An empty window appears with axes and grid points. There are two grids, the minor and the major. In this tutorial the minor grid points are 1 µ apart and the major grid point occurs once every 5 µ.

Choose **Create → Instance** from the menu bar or type: **I**, or hit the **Instance** button on the left side. Select **Browse** and select library **sigehp → FET** (if **Show Categories** selected) **→ pfet → layout**.

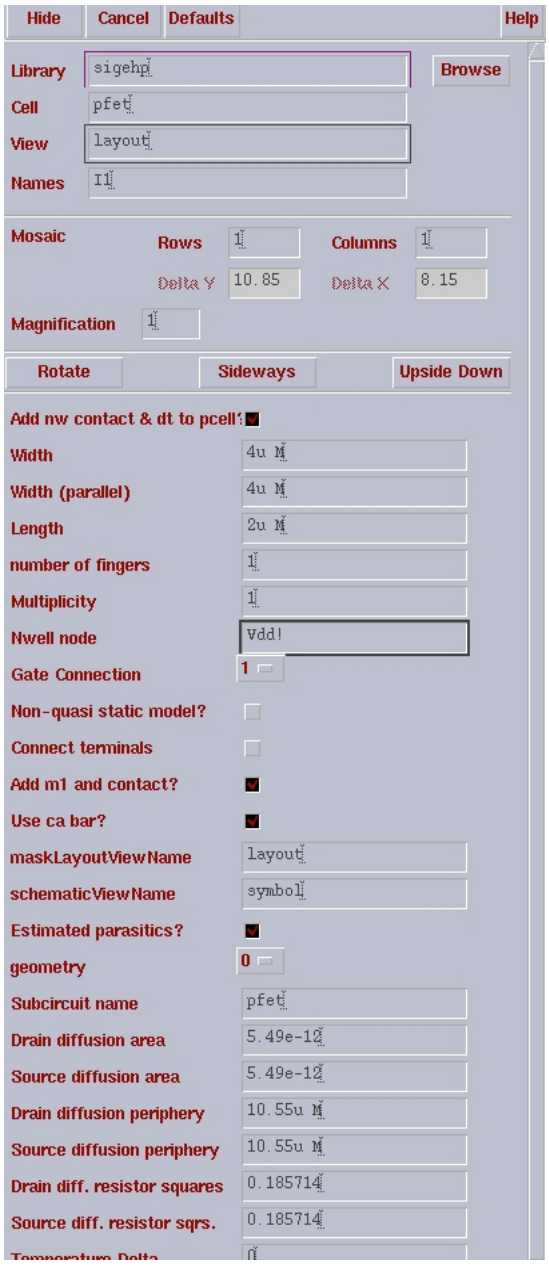


Figure 6.3: The Create Instance Form

Fill in the correct width, length and nwell node values. Notice what happens to the device outline when you select the **'Add nw contact & dt to pcell?'**. Move the cursor into the *Cellview* window; outlines of the shapes in the pfet follow the cursor. Left click at the desired position of the instance in the *Cellview* window. If you move your cursor to the right you will find the outline of another identical transistor attached to it because the create instance command is a repeating command. To stop the command, press the **Esc** key.

Now create the nfet instance. If you cannot see both transistors, adjust the view by pressing the **f** key or choosing **Window → Fit all** from the menu banner.

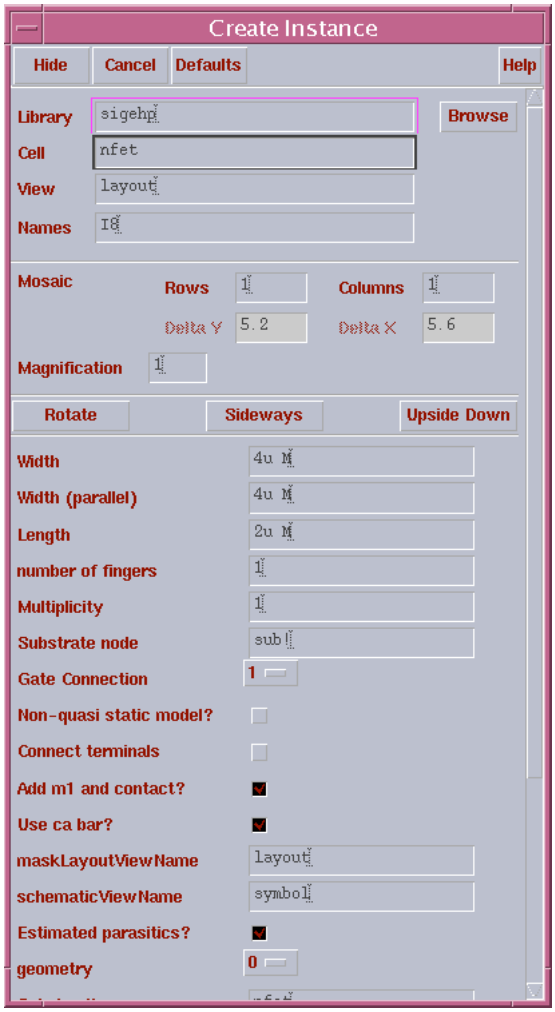
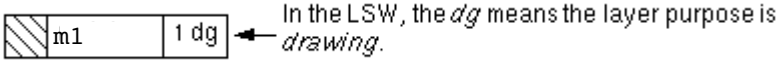


Figure 6.4: The Create Instance Form for an nFET

To create the connections necessary to build an inverter, you must choose the correct entry layer. Every layer in the LSW has a purpose, such as net or drawing. Most layouts use the layer whose purpose is drawing. (The abbreviation dg means drawing)



Click the **m1 dg** layer in the LSW (Remember to use only the left mouse button. Any other button will change the visibility of the layer).

The m1 layer is outlined in bold and appears at the top of the LSW window telling you that this is the current entry layer. Now choose **Create → Rectangle** from the menu bar. The prompt at the bottom of the layout window reads, **"Point at the first corner of the rectangle"**.

The prompt at the bottom of the layout window now reads, **"Point at the opposite corner of the rectangle"**. A rectangle appears and stretches as you move the cursor. You are still in the rectangle mode because the **create** command is a repeating command.

To connect the inputs you must use poly (pc). So, click the **pc dg** layer in the LSW with the mouse. The pc layer appears at the top of the LSW. Choose **Create → Path** from the menu banner. The *Create Path* form appears. In the form the width is set to 0.5 µ, which is the minimum width for the poly1 layer. The prompt at the bottom of the layout window reads, **"Point at the first corner of the rectangle"**. The path appears. The double-click tells the system where to end the path. Try and make your layout look like the example.

Instance the m1-lm connections (included in the library VLSI_CLASS) as in the example layout. Place the m1-lm instance as shown in the example.

Now, you need to create the input and output pins and the jumper pins. Use the Create Pin menu option and choose the appropriate pin attributes and assign labels to the pins. At this point, the layout would look like this:

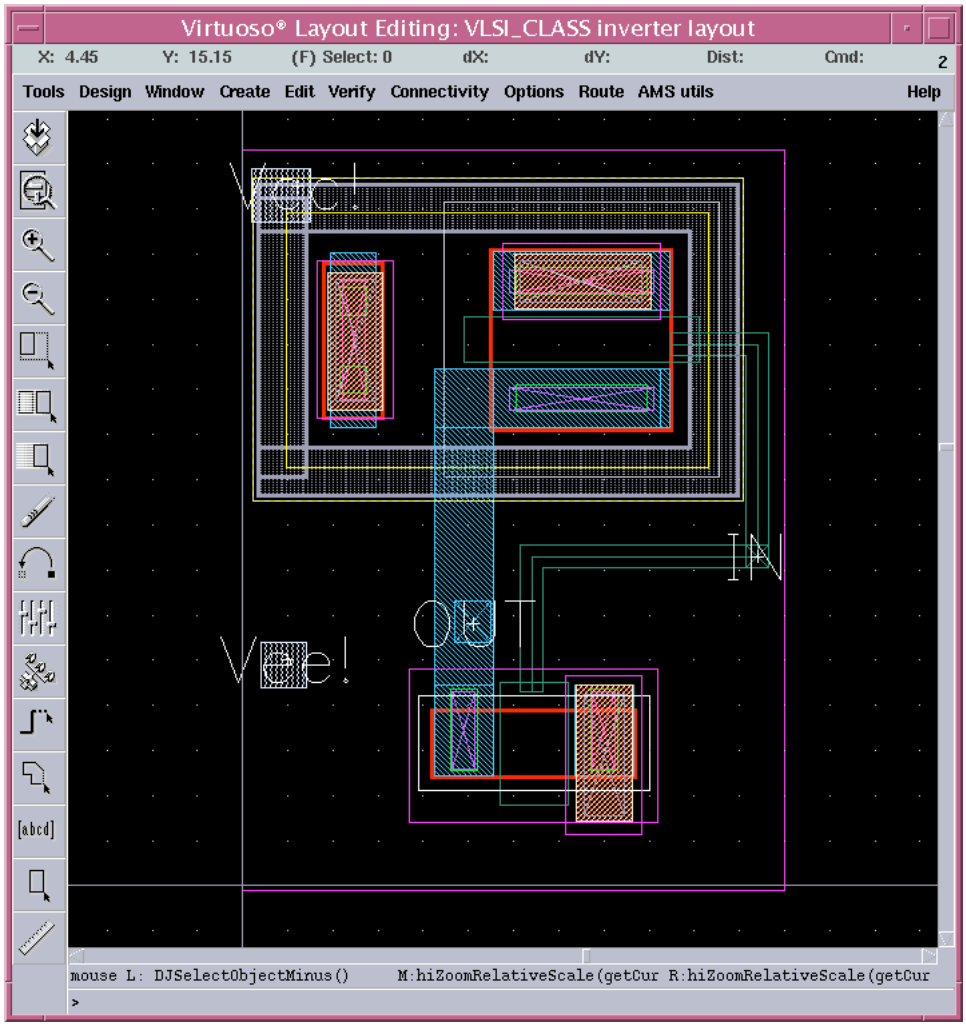


Figure 6.5: Layout with Poly Connection

The LM layers have to be created now. Look at the example layout of the inverter in the VLSI_CLASS library to create the layers. Choose “lm dg” in the LSW window by clicking the left mouse button on it and use the “Create Rectangle” option.

6.3 Checking Design Rules and Correcting Errors

To perform design rule checking, choose **AMS Utils → Checking → DRC**. The *DRC* (Design Rule Checker) form appears. Click **OK** to start the design rule checker. If you followed the previous steps exactly, you should have no errors (if you are really lucky). The error messages appear in the CIW. Note that the five level metal and sigehp Diva check are not really errors, so the least number of errors you can have is 2. If you have errors, note the message, and if you cannot id where they are, you can go to the LSW window, select **marker er** (near the bottom), select **NV** (none visible) and then redraw the window. Zoom in on the white boxes, redraw with **AV** and figure out what is wrong. Liberal use of the ruler is a good idea when starting out. You can left click on the error boxes and the message should appear again in the CIW window. You may also view the bad features one by one using **AMS Utils → Checking → Marker Select**. This form allows you to select a particular error from the list **‘Pick a message to checkout’** and the window zooms to that marker when **Apply** is selected. Also **Verify → Markers → Explain** may be used to explain a particular error.

The easiest way to correct errors is to delete the objects causing them and recreate them using the instructions in this tutorial. To delete objects, select the shape you want to delete by clicking on it with the left mouse button and hit the **Del**(ete) key.

Once you are error free, save your design by choosing **Design → Save** from the menu bar.

To exit, in the CIW choose **File → Exit** to exit from the Cadence design system.

6.4 Making Larger Layouts

Open the NAND gate example and notice that the 2 pfets are in the same nwell. You should put all the pfets in the same nwell to save space. This is done using the following steps:

1. Instance a pfet with nwell and dt as normal.
2. Select **Edit → Hierarchy → Flatten**
3. Select '**Flatten Mode**' to **displayed levels**, and select (check) both '**Flatten Pcells**' and '**Preserve Pin**'.
4. Select the pfet (box it) and hit **Del**.
5. Stretch the dt and nw to fit the number of transistors. You can have a very large number of pfets, as long as there is an nwell contact every 150 μ.
6. Pointers: the dt should be 1.1 μ wide. The nw and ns boxes are on top of each other and they should be 0.45 μ away from the inside edge of the dt. The white bounding box should be just outside the dt box. You should be familiar with the following shortcut keys:

g turn on/off gravity:	Gravity will snap lines to the nearest edge (turn this off).
s stretch:	The yellow outline tells you what layer will be stretched.
Control D:	Deselects the last thing selected.

7. After stretching, instance pfets without the dt & nw option. Make sure they align correctly, otherwise you will have many DRC errors.

Chapter 7: Layout Verification

7.1 Layout vs. Schematic Check (LVS)

Your final step is LVS. The purpose of LVS is to see if the layout you created matches the schematic. It must match in pin names, direction, logical nets, and circuit parameters. A netlist from your schematic is compared to a netlist from your layout. LVS looks for matches or mismatches. In order to do this, Cadence draws a schematic from your layout. This is called the extracted view. To create an extracted view, from the layout window select **AMS Utils → Checking → Extract** and click **OK**. Go into the library manger and open up the extracted view. Zoom in and verify that Cadence has drawn circuit symbols on top of your layout. You can turn off the lm layer by middle clicking **lm (nt)**. Do not do any drawing on the extracted view.

Now open up the schematic view and select **Tools → Diva, Verify → LVS.**

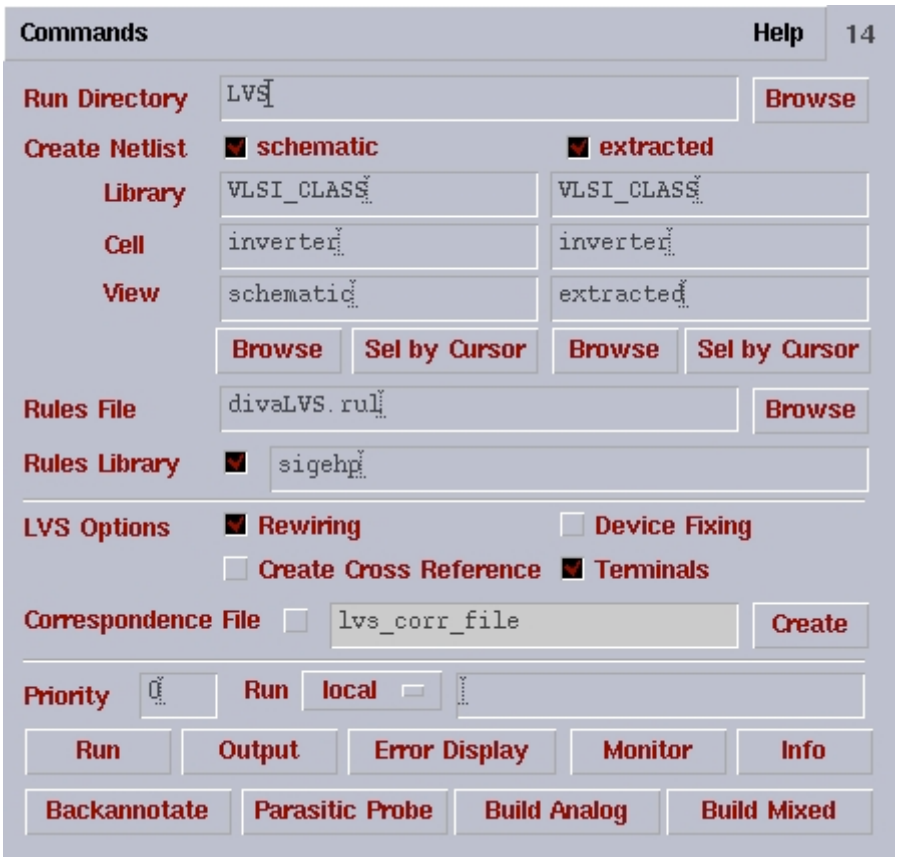


Figure 7.1: LVS Window

Fill in the boxes as shown and insure the rewiring is not selected. Then hit **Run**. In your CIW the message **"the LVS job is now started..."** should appear. Later a window will appear hopefully saying the LVS job completed successfully. In either case, hit **Output** and read the messages. What you want to see is that **"the net lists match."** Of course this is not likely to occur on the first go-around.

Cadence has a nice tool to isolate problems. One wrong wire could cause 5 - 10 error messages, so do not become discouraged. Open up the extracted view and then hit **error display** in the LVS window.

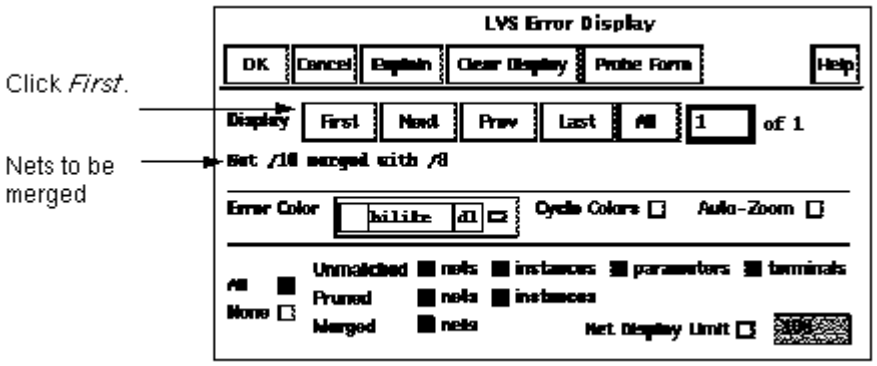


Figure 7.2: Error Display

The error display will take you through all of your errors. First select none in the error display window, then unmatched nets and terminals. Then click on **Auto-zoom** and then hit **First** and examine the error. Nets are highlighted in yellow. Check against you schematic and layout and see what is wrong. Often a missing via or wrong connection or wrong pin is the most common mistake. Also note that the number of transistors should match in the schematic and layout. If they are different your layout will never pass LVS.

Once you have look through the errors, using **Next**, change your layout, do another extraction, and repeat the process again until error free. Congratulations, you now can go on to larger designs.

7.2 Printing out Your Results

Again you can save your results using the xv screen capture program. Type: **xv &** to start the program at the Unix prompt. Click on the xv window with the right mouse button and select **Grab** in the new window. Under **AutoGrab**, use 2 sec and use the middle mouse button to box the image you want to save. An image window should pop up. Under **Windows → Color Editor**, you can change black to white (swing the dials to 255, 255, 255) and turn the saturation up. You can also click on the other colors to customize the image. Save the image as a **gif-full color**.

Chapter 8: Verilog

This chapter has been prepared to help you get started with Verilog. The examples in sections 8.2, 8.3 and 8.4 have been taken from the book by J. Bhasker [1]. The examples in section 8.4 have also been taken from the Cadence Openbook Manual [2]. The author claims no credit for them. To learn more about Verilog you must refer to [1] or any standard textbook on the subject.

8.1 A Quick Start Verilog Tutorial

NOTE: Before running the Verilog tool, make sure you copy the 'verilog' file from the /cadhome/cadtest directory to your home directory. If you don't do this correctly, another version of Verilog will execute that is very similar to this one except that plotting will be disabled due to a licensing issue.

The basic unit in a Verilog description is a *module*. A *module* describes

- The structure or functionality of the design
- The ports through which it communicates with other modules

The structure of a design is described using switch-level primitives, gate-level primitives or user-defined primitives. The data-flow is described using continuous assignment statements and the sequential behavior is modeled using procedural constructs. The syntax of a module is as follows;

module *module_name (port_list);*
Declarations:
reg, wire, parameter,
input, output, inout,
function, task…..
Statements:
Initial statement
Always statement
Module instantiation
Gate instantiation
UDP instantiation
Continuous assignment
endmodule

Declarations are used to define items like registers and parameters that are used within the module. Statements are used to define the functionality of the design. Declarations and statements may be made anywhere in the design with the restriction that an item is declared before it is used. However, it is good design practice to put all declarations before any statements.

8.1.1 Modeling a half-adder circuit

```
module HALF_ADDER (A, B, SUM, CARRY);
        input A, B;
        output SUM, CARRY;

        assign #2 SUM = A ^ B;
        assign #5 CARRY = A & B;
endmodule
```

The name of the module is *HALF_ADDER*. It has four ports, two input and two output ports. The input ports are *A* and *B*. The output ports are *SUM* and *CARRY*. The ports are 1-bit ports since no bit range has been specified. Also, the ports are of the **net** datatype since no declaration has been specified explicitly. Two continuous assignment statements describe the dataflow in the module. The order in which the statements

appear is not important. Execution of each statement is based on events occurring on nets *A* and *B*. In the previous example, '&' indicates a logical AND operation and '^' a logical XOR operation.

A design can be described in any of the following styles:
1. Dataflow style
2. Behavioral style
3. Structural style
4. Mixed style

8.1.2 Specifying delays in Verilog

Delays in Verilog models are specified in terms of time units. The continuous assignment statements in the half-adder circuit described have delays specified in them.

```
assign #2 SUM = A ^ B;
```

The *#2* refers to 2 time units.

The time unit is associated with physical time through a compiler directive **`timescale**. Such a directive must be specified before the module declaration. For example

```
`timescale 1ns/100ps
module HALF_ADDER (A, B, SUM, CARRY);
. . .
endmodule
```

This says that one time unit is to be treated as 1 ns and that the time precision is to be 100 ps. Thus in the half_adder example, #2 refers to 2 ns. When no such compiler directive is specified, the default specified in the IEEE Verilog HDL Standard is used.

8.2 Using the Dataflow Style

In the dataflow style, continuous assignment statements are used to model the design. In a continuous assignment statement, a value is assigned to a net. The syntax is:

```
assign [delay] LHS_net = RHS_expression;
```

Whenever the value of the right-hand side expression changes, the new value is assigned to the left-hand side net after the specified delay. If no delay is specified, the default is zero delay.

8.2.1 A 2-to-4 decoder circuit using dataflow description style

```
`timescale 1ns/1ns
module DECODER2x4(A, B, EN, Z);
        input A, B, EN;
        output [0:3] Z;
        wire ABAR, BBAR;

        assign #1 ABAR = ~A;                        // line 1
        assign #1 BBAR = ~B;                        // line 2
        assign #2 Z[0] = ~ (ABAR & BBAR & EN);      // line 3
        assign #2 Z[1] = ~ (ABAR & B & EN);         // line 4
        assign #2 Z[2] = ~ (A & BBAR & EN);         // line 5
        assign #2 Z[3] = ~ (A & B & EN);            // line 6
endmodule
```

The first statement is the compiler directive that sets the time unit for all modules. In this example the time delay unit is 1 ns and the time precision is 1 ns. The module *DECODER2x4* has 3 input ports *A*, *B*, *EN*. The declaration

wire *ABAR, BBAR;*

declares the two wires *ABAR* and *BBAR*. The module also has 6 continuous assignment statements that model the behavior of the circuit. Remember that:

1. The #1 represents a delay value.
2. The continuous assignment statements execute concurrently i.e., they are order independent.

This behavior of the circuit can be explained as follows: whenever *EN* changes, the lines 3-6 are executed since they depend on the value of the *EN* signal (*EN* occurs in the expression on the RHS of the continuous assignment statements). Thus 2 ns after *EN* changes, the outputs *Z[0-3]* get assigned new values.

Whenever *A* changes, the lines 1, 5 and 6 are executed since they depend on the value of A. Thus *ABAR* gets assigned a new value 1 ns after A changes and, the outputs *Z[2]* and *Z[3]* get assigned new values 2 ns after *A* changes. The change in the value of *ABAR* triggers the execution of lines 2 and 3. Thus, 2 ns after *ABAR* changes, *Z[0]* and *Z[1]* get assigned new values.

8.3 Using the Behavioral Style

In the behavioral style, the behavior of the design is described using procedural constructs. These are:

Initial statement: This statement is executed only once.

Always statement: This statement is executed repeatedly.

All initial and always statements begin execution at time zero concurrently. In these statements, values can be assigned to the **register** datatype only. This datatype has the characteristic that it retains its value until a new value is assigned to it.

8.3.1 A 1-bit full-adder using behavioral description style

```
module FA_SEQ(A, B,CIN,SUM, COUT);
input A, B, CIN;
output SUM, COUT;
reg SUM, COUT;
reg T1, T2, T3;

always
        @(A or B or CIN) begin
        SUM = (A ^ B) ^ CIN;
        T1 = A & CIN;
        T2 = B & CIN;
        T3 = A & B;
        COUT = (T1 | T2 | T3);
        end
endmodule
```

The module *FA_SEQ* has three inputs *A*, *B*, *CIN* and two outputs *SUM*, *COUT*. There are three components of the datatype **reg**: *T1*, *T2*, *T3*. These have to be of the type **reg** since they are assigned values within the **always** statement.

The **always** statement consists of a sequential block (**begin-end** pair) associated with an event control (the expression following the @ symbol). In this example this means that whenever an event occurs on *A*, *B* or *CIN*, the sequential block is executed. Within the **begin-end** pair the statements are executed sequentially

and the execution is suspended after the last statement has been executed. After the sequential block completes execution, the **always** statement waits for an event to occur on *A*, *B* or *CIN*.

The statements within the sequential are examples of blocking procedural assignments. A blocking procedural assignment completes execution before the next one executes. It may have a delay associated with it. A delay may be

- inter statement delay: The delay by which the execution of the statement is delayed. For example: #4 *T1 = A & CIN*;
- intra statement delay: The delay between computing the value on the RHS of an assignment statement and the assignment of the value to the LHS. For example: *SUM = #3 (A ^ B) ^ C*;

8.3.2 A module using the initial statement

```
`timescale 1ns/1ns
module TEST(POP, PID);
        output SUM, COUT;
        reg POP, PID;

initial
        begin
          POP = 0;
          PID = 0;
          POP = #5 1;
          PID = #3 1;
          POP = #6 0;
          PID = #2 0;
        end
endmodule
```

The initial statement contains a sequential block that starts execution at time 0ns and it suspends forever after executing the last statement within the **begin-end** block.

8.4 Using the Structural Style

Structures in Verilog can be built using

1. Built-In gate primitives (at the gate level)
2. Switch-level primitives (at the transistor level)
3. User-defined primitives (at the gate level)
4. Module instances (to create hierarchy)

8.4.1 A 1-bit full-adder in structural description style using built-in gate primitives

```
module FA_STR(A, B,CIN,SUM, COUT);
        input A, B, CIN;
        output SUM, COUT;
        wire S1, T1, T2, T3;

        xor
        X1 (S1, A, B),
        X2 (SUM, S1, CIN);
        and
        A1 (T3, A, B),
        A2 (T2, B, CIN),
        A2 (T1, A, CIN);
        or
        O1 (COUT, T1, T2, T3);
endmodule
```

This module uses the built-in gates **xor**, **and,** and **or**. Wires *S1, T1, T2, T3*, connect these. The order in which the gate instances appear does not matter since no sequential evaluation order is implied.

8.4.2 4-bit full-adder in structural description style using module instantiation

```
module FOUR_BIT_FA(FA, FB, FCIN, FSUM, FCOUT);
        parameter SIZE = 4;
        input [SIZE:1] FA, FB;
        output [SIZE:1] FSUM;
        input FCIN;
        output FCOUT;
        wire [1:SIZE-1] FTEMP;

        FA_STR
                FA1(.A(FA[1]), .B(FB[1]), .CIN(FCIN),
                        .SUM(FSUM[1]), .COUT(FTEMP[1])),
                FA2(.A(FA[2]), .B(FB[2]), .CIN(TEMP[1]),
                        .SUM(FSUM[2]), .COUT(FTEMP[2])),
        FA3(FA[3]), FB[3]), FTEMP[2], FSUM[3], FTEMP[3]),
        FA4(FA[4]), FB[4]), FTEMP[3], FSUM[4], FCOUT);

endmodule
```

In this example, a 4-bit full adder is made by instantiating four 1-bit modules. In a module instantiation, the ports can be associated by name or by position. *FA1* and *FA2* use name associations while *FA3* and *FA4* use positional associations. In named associations, the name of the port and the net to which it is connected are explicitly described using the syntax:

.port_name (net_name)

In positional association, the order of association is important. For this example, in the instance *FA4*, the first one *FA[4]* is connected to port *A* of *FA_STR*, the second one *FB[4]* is connected to port B of *FA_STR*.

8.5 Verifying a Design

You can verify your design by simulating it with Verilog-XL. To verify that the two-bit adder works, it must be driven and its results observed. This is done by modeling a test fixture and connecting the module under test to it. In this example, a behavioral 2-bit adder is modeled. Then the test fixture to test this model is created.

```
// Behavioral 2-bit adder
module B_TWO_BIT_ADDER (SUM, C_OUT, A, B, C_IN);
        input [1:0] A, B;
        input C_IN;
        output [1:0] SUM;
        output C_OUT;
        assign #200 {C_OUT, SUM} = A + B + C_IN;
endmodule
```

1. Enclose the model with **module** and **endmodule** keywords. Assign the model a name, and enclose the primary inputs and outputs in parentheses. The file /cadhome/cadtest/2bit_adder.v contains the module named *B_TWO_BIT_ADDER* above.
2. Specify the names and the sizes of the inputs and outputs.
3. Add the three inputs and assign the result to a three-bit wire, which is a concatenation of two wires, *C_OUT* and *SUM*. The concatenation operator **{}** tells Verilog to make *C_OUT* the high order bit of the three bit wire. *SUM* is the low order bit. Since the **assign** keyword is used, *C_OUT* is updated each time there is a change in any one of the inputs.

The following test fixture for this module is called *TESTER* and is in the file /cadhome/cadtest/test_fixture.v.

```
module TESTER;
reg [1:0] BUS_A, BUS_B;
reg C_IN;
wire [1:0] SUM;
wire C_OUT;
B_TWO_BIT_ADDER UNDER_TEST (SUM, C_OUT, BUS_A, BUS_B, C_IN);
    initial
        begin
            #300 C_IN = 1'b0;
            BUS_A = 2'b00;
            BUS_B = 2'b01;
        end
    always
        begin
            #100;
            $display("at time %0d bus_a = %b, bus_b = %b, c_in =%b, sum = %b, c_out = %b",
                $time, BUS_A, BUS_B, C_IN, SUM, C_OUT);
            (if $time > 1000) C_IN = 1'b1;
            (if $time > 2000) $stop;
        end
endmodule
```

It uses registers to store values, and then connects the adder to those registers. It assigns initial values to those registers by using an **initial** statement. The example monitors the system value **$time**.

1. Use the **module** and **endmodule** keywords to enclose the *TESTER* module. *TESTER* has no fixed inputs or outputs.
2. Declare the registers to drive the inputs and outputs.
3. Instantiate the behavioral 2-bit adder by specifying the module name, instance name, and inputs and outputs.
4. Now specify your input stimuli. In this case a non-repeating stimuli have been placed in the **initial** block. After a delay of 300 time units, 1-bit binary 0 is assigned to *C_IN* and a 2-bit binary 0 is assigned to *BUS_A*. Within the **always** block a delay of 100 time units has been included along with the display statement.

The syntax of the **$display** statement is similar to the C language printf function. The time value is in decimal and the zero in *%0d* specifies that the value be displayed in minimum amount of space.

Assume that you placed your behavioral 2-bit adder in the file 2bit_adder.v and your test module in the file test_fixture.v. To simulate your design, at the %: prompt type the following:

```
$> ~/verilog +gui 2bit_adder.v test_fixture.v
```

{This is what you see (after some introductory information) in the new *Cadence Verilog-XL* window:}

```
Compiling source file "2bit_adder.v"
Compiling source file "test_fixture.v"
Highest level modules:
TESTER

at time 100 bus_a=xx, bus_b=xx, c_in=x, sum=xx, c_out=x
at time 200 . . .
.
.
.
```

In the *Cadence Verilog-XL* window, **left-click** on **'Select → Signals'**. This will highlight all the I/O variables in the program. **Left-click** on **'Control → Reset Simulation'**. **Left-click** on the rightmost button to the right of **'Tools:'** or **left-click** on **'Tools → Waveforms...'** to open up the *DAI Signalscan Waveform:1* window. In the *Cadence Verilog-XL* window **left-click** on the button with the large **'▶'** or **left-click** on **'Control → Run'** to begin the simulation. The waveform will be plotted in the *DAI Signalscan Waveform:1* window.

NOTE:

You **must** type:

```
~/verilog +gui 2bit_adder.v test_fixture.v
```

This has to be done exactly as written, whether you think you know what's going on or not! If you type:

```
verilog +gui 2bit_adder.v test_fixture.v
```

a different version of the program starts to execute, one for which there are no licenses, and you will end up with errors and no output. The default pathnames are set up for Cadence icfb, which is incompatible with Cadence Verilog. The '`~/`' forces the execution of a script in your home directory needed to change your environment to run the correct version of Verilog.

References

[1] J. Bhasker, "*A VERILOG HDL Primer*", Star Galaxy Press, PA, 1997

[2] Cadence Openbook Manual, Cadence

Chapter 9: VHDL

This chapter has been prepared to help you get started with VHDL using Cadence AFFIRMA NCVHDL in the RPI ECSE domain (by Eric Thompson).

9.1 A Quick Start VHDL Tutorial

The basic units in a VHDL description are an *entity* and *architecture*. These describe:

- The structure or functionality of the design
- The ports through which it communicates with other modules

The structure of a design is described using switch-level primitives, gate-level primitives or user-defined primitives. The data-flow is described using assignment statements and the sequential behavior is modeled using procedural constructs. The syntax of a VHDL program is as follows:

```
library
use

entity entity_name is
    port(
        port_list:
            in, out, inout, buffer, linkage
            std_logic, std_ulogic, Boolean, string, integer, float
    );
statements:
end entity_name;

architecture arch_name of entity_name is
    component comp_name
        port(
            port_list:
        )
    end component;
    signal assignments:
begin
        architecture statements:
end arch_name;

configuration config_name of entity_name is
    configuration declarations;
     block configurations;
end configuration;

package package_name is
    package declarations;
    package body package_body_name is
        package body declarations;
    end package body;
end package;
```

Declarations are used to define items like registers and parameters that are used within the entity. Statements are used to define the functionality of the design. The configuration block and the package block are both optional.

9.1.1 Modeling a one-bit adder circuit

The following is a 1-bit adder example that can be found at /cadhome/cadtest/VHDL/1bit_adder.vhd.

```
library IEEE;

use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity ONE_BIT_ADDER is
        port(
                A               : in std_logic;
                B               : in std_logic;
                CARRY_IN        : in std_logic;

                SUM             : out std_logic;
                CARRY_OUT       : out std_logic;
        );
end ONE_BIT_ADDER;

architecture ONE_BIT_ADDER_RTL of ONE_BIT_ADDER is
begin
        SUM     <= A xor B xor CARRY_IN after 2 ns;
        CARRY_OUT       <= (A and B) or (A and CARRY_IN) or (B and CARRY_IN) after 2 ns;
end ONE_BIT_ADDER_RTL;
----------------------------------End of File-------------------------------
```

The name of the entity is *ONE_BIT_ADDER*. It has five ports, three input and two output ports. The input ports are *A, B,* and *CARRY_IN*. The output ports are *SUM* and *CARRY_OUT*. The ports are 1-bit ports since no bit range has been specified. Also, the ports are of the **std_logic** datatype as the declaration has specified explicitly. Two continuous assignment statements describe the dataflow in the module. The order in which the statements appear is not important. Execution of each statement is based on events occurring on nets *A*, *B*, and *CARRY_IN*. In this example, 'and' indicates a logical AND operation, 'or' indicates a logical OR operation and 'xor' a logical XOR operation.

9.1.2 Creating a test bench for the one-bit adder

In order to simulate the one-bit adder design, a VHDL test bench must be created similar to the test fixture created for the Verilog simulation. The following program in the file test_bench.vhd provides the necessary commands needed to test the 1-bit adder example 1bit_adder.vhd given previously. This file is located at /cadhome/cadtest/VHDL/test_bench.vhd.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
--use worklib.all;

entity TESTER is
end TESTER;

architecture TESTER_ARCH of TESTER is

	component ONE_BIT_ADDER
		port(
			A : in std_logic;
			B : in std_logic;
			CARRY_IN : in std_logic;

			SUM : out std_logic;
			CARRY_OUT : out std_logic
		);
	end component;

	signal A : std_logic;
	signal B : std_logic;
	signal CARRY_IN : std_logic;
	signal SUM : std_logic;
	signal CARRY_OUT : std_logic;

begin
	UUT : ONE_BIT_ADDER
		port map(
			A => A,
			B => B,
			CARRY_IN => CARRY_IN,
			SUM => SUM,
			CARRY_OUT => CARRY_OUT
		);

--Add your stimulus here…

	A <= '1' after 10 ns, '0' after 100 ns, '1' after 250 ns;
	B <= '1' after 25 ns, '0' after 50 ns, '1' after 80 ns, '0' after 300 ns;
	CARRY_IN <= '0' after 5 ns, '1' after 75 ns, '0' after 120 ns, '1' after 200 ns;

end TESTER_ARCH;

--configuration cfg_bench of tb_arch is
--	for tb_arch
--		for UUT : trial1
--			use entity worklib.trial1(trial1);
--		end for;
--	end for;
--end cfg_bench;
```

9.1.3 Specifying delays in VHDL

Delays in VHDL models are specified in terms of seconds. The assignment statements in the adder circuit described have delays specified in them for the outputs SUM and CARRY_OUT of 2 ns.

SUM <= *A* **xor** *B* **xor** *CARRY_IN* **after** *2 ns*;
CARRY_OUT <= (*A* **and** *B)* **or** *(A* **and** *CARRY_IN)* **or** *(B* **and** *CARRY_IN)* **after** *2 ns*;

The time duration for a simulation is specified in the NC simulator using the run command, i.e. to specify a 500 ns run, type:

```
ncsim> run 500 ns
```

9.2 Setting up for VHDL

Step 1: Create a folder for the design and go into that folder.

All commands should be run from inside this directory. If a design directory exists, use that. If not, one must specify where to find one's designs and its libraries, etc. After creating a directory in your account for the VHDL files, create another inside it called worklib for work libraries:

```
$> mkdir [folder_name]        {i.e. $> mkdir VHDL}
$> cd [folder_name]           {i.e. $> cd VHDL}
&> mkdir [work_library_name]  {i.e. $> mkdir worklib}
```

All compiled and elaborated design files end up in the VHDL directory and may be used for all projects. This will be specified in the cds.lib and hdl.var files (remember what you called it). The two other setup files, **cds.lib** and **hdl.var**, listed below, specify the path to specific libraries and also the user *work* library. In this case the user *work* library has been called *worklib* but it may be given any name.

hdl.var

```
SOFTINCLUDE $CDS_INST_DIR/tools/inca/files/hdl.var
DEFINE USE_NEW_SIMWAVE_WINDOW ON
DEFINE WORK worklib
```

cds.lib

```
SOFTINCLUDE $CDS_INST_DIR/tools/inca/files/cds.lib
DEFINE worklib ./worklib
```

Step 2: Set up the environment.

To start using the Cadence tools one needs to tell Cadence where its license, executables and libraries are. The script the author uses is called rc.vhdl, listed below and can also be found at /cad/rc_scripts/rc.vhdl.

rc.vhdl

```
#  This is a Cadence RC file
#  This file should be sourced from your ~/.cshrc file if
#  you want to run Cadence Affirma NC VHDL simulator
#
#  NOTE: THIS IS NOT COMPATIBLE WITH Cadence IC 4.4.X
#        ENVIRONMENTAL VARIABLES NEEDED BY IC ARE REDIFINED HERE!
####
#  last modified by erict 6/4/01
#  (update for ldv3.0)

setenv CDS_INST_DIR /cad/cds/ldv3.0

setenv CDS_LIC_FILE /cad/license/license.dat

setenv LD_LIBRARY_PATH
${LD_LIBRARY_PATH}:/usr/dt/lib:$CDS_INST_DIR/tools/lib:$CDS_INST_DIR/tools/inca/lib

if ( "$PATH" !~ *"$CDS_INST_DIR/tools.sun4v/bin"* ) then
    set path=($path $CDS_INST_DIR/tools.sun4v/bin)
endif

#  end
```

One may either source this file by hand via:

```
$> source /cad/rc_scripts/rc.vhdl
```

or one may put the above command in one's .cshrc file to be sourced automatically. **NOTE: This rc file may cause conflicts with other Cadence tools. Sourcing `rc.vhdl` last may prevent `icfb` and `hspice` from working while sourcing `rc.hspice` last may prevent `vhdl` from working. Source (again) the appropriate script before running `vhdl` or `hspice` if there are any problems.**

9.3 Analyze and Elaborate the Design.

All design and test bench files need to be analyzed and elaborated before they can be simulated. The analyzer, *ncvhdl*, encapsulates two tools: the VHDL parser and the code generator. An example test file, /cadhome/cadtest/CDS/LDV3.0/test.vhd, and its test bench, /cadhome/cadtest/CDS/LDV3.0/test_bench.vhd, are available for testing purposes. They use the IEEE STD libraries.

The parser performs syntactic and static semantic checking on the input source files. If no errors are found, compilation produces an internal representation for each HDL design unit in the source files. These intermediate objects are stored in a library directory (the worklib specified above).

The elaborator, *ncelab*, constructs a design hierarchy based on the information in the design, establishing signal connectivity, and computes initial values for all objects in the design. The elaborated design hierarchy is stored in a simulation snapshot file, which is used by the simulator.

9.3.1 Analyzing the design

The generic command for the analyzer in NCVHDL is:

```
$> ncvhdl [OPTION] [FILE]
```
(*Note:* Use: "`ncvhdl -help`" for a complete list of available options).

For more informational messages and to use VHDL93, use:

```
$> ncvhdl -messages -v93 [FILE]
```

9.3.2 Elaborating the design

The generic command for the elaborator in NCVHDL is:

```
$> ncelab [OPTION] [lib.]cell[:view]
```
(*Note:* use: "`ncelab -help`" for a complete list of available options).

where "lib" is the library where the design is kept, "cell" is the name of the design entity, and "view" is the name of the design architecture.

9.4 Simulating

One may now simulate the design to help determine whether it is functionally correct. The generic command to simulate a snapshot of in NCVHDL is:

```
$> ncsim [OPTION] [lib.]cell[:snap]
```
(*Note:* use: "`ncsim -help`" for a complete list of available options).

Again, "lib" is the library where the design is kept, "cell" is the name of the design entity, and "snap" is the name of the snapshot (there can be multiple).

If one would like to view the simulation in a GUI, specify the -gui option:

```
$> ncsim -gui [cell]
```

One will then see a simulator window as shown below in figure 9.1.

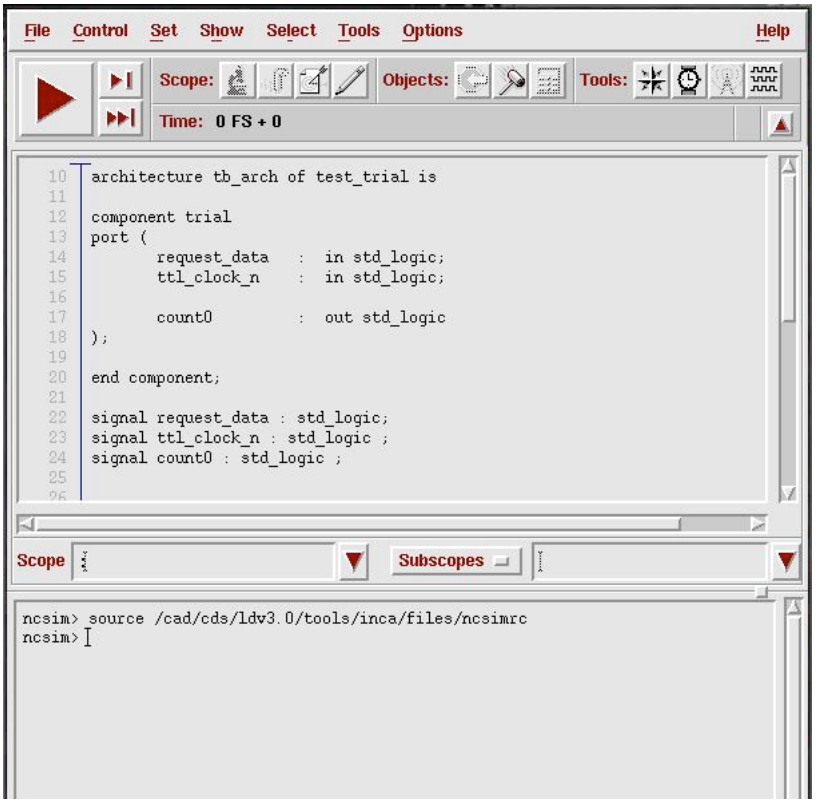


Figure 9.1: The Cadence Affirma NC VHDL Simulator Window

The window in figure 9.1 displays the architecture of the test_bench.vhd file. The menus of this tool are quite extensive and provide the user with a lot of debugging facilities like setting breakpoints and watch values. The lower portion of the window gives a command prompt (`ncsim>`). This allows you to type a command rather than choosing options from the menu.

9.4.1 Starting the DAI Signalscan tool

Before invoking Signalscan, it is important to select the signals as in figure 9.2 to be displayed in the waveform window. Choose **Select → Signals** from the ncsim menu bar (refer to figure 9.1). NCSIM will then select all allowed signals and the window will appear similar to figure 9.2.

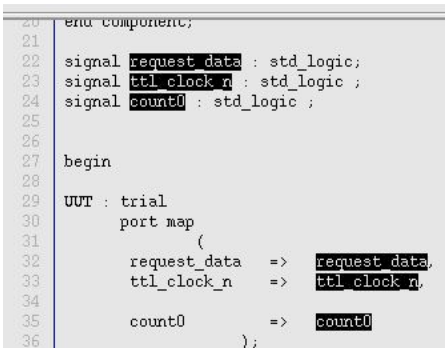


Figure 9.2: Selected signals in the Cadence Affirma NC VHDL Window

To invoke the signal scan click on the  button in the upper right corner (see figure 9.1). The *DAI Signalscan* window with the signals selected will open (figure 9.3):

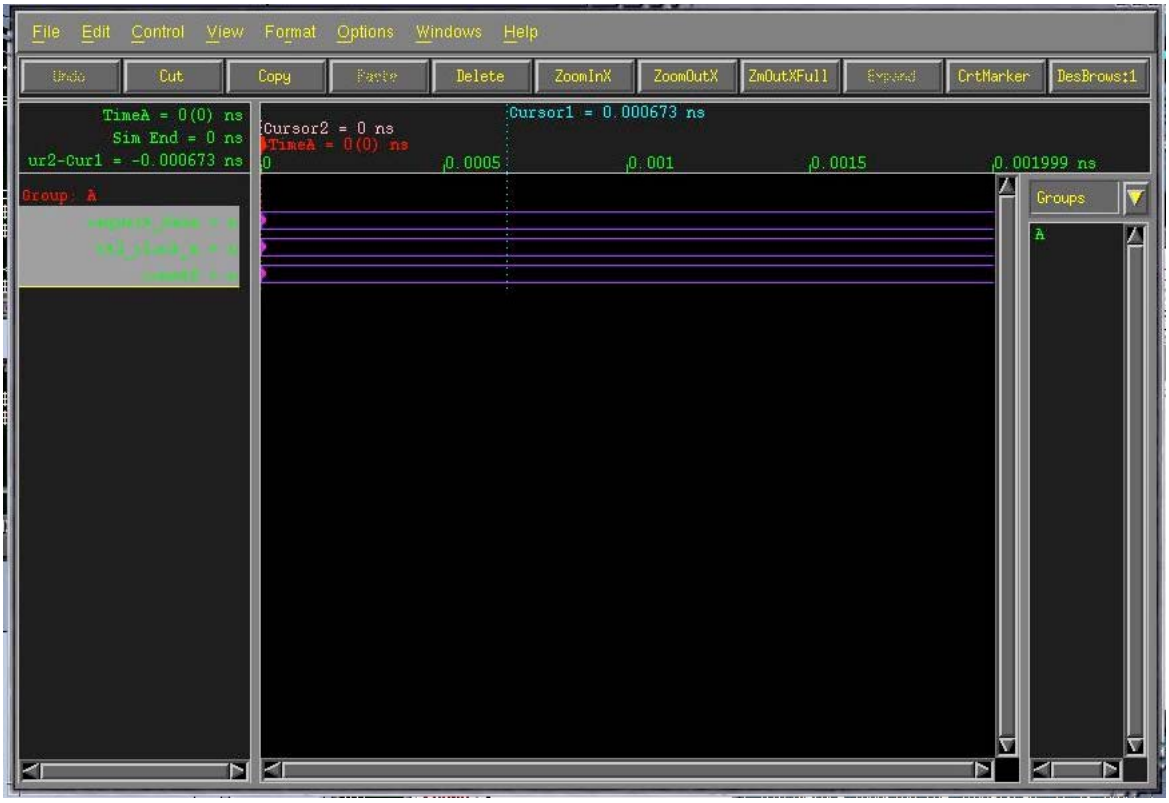


Figure 9.3: The DAI Signalscan Window

9.4.2 Running the Simulation

To start the simulation click on that "run" button,  , in the *simulator* window. This will run the simulation to the end of the test vectors provided in the test_bench.vhd file. There are run options to single step through or over the code. Another nice way of running a simulation is to specify the absolute or relative time of a run. The following example runs the simulation for 500 ns:

```
ncsim> run 500 ns
```

After having completed the simulation the DAI Signalscan displays the waveform of the individual signals (figure 9.4, window shown after clicking on **ZmOutXFull**).

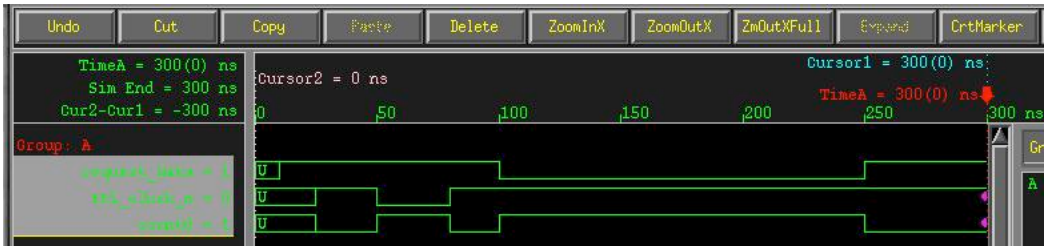


Figure 9.4: DAI Signalscan Waveform Window

More information about the Cadence tools is available by either browsing through the HELP menu or by typing:

```
$> openbook
```

on the Unix command prompt. This opens the Cadence OpenBook for the entire suite of tools available under the LDV 3.0 release.

9.5 Summary

These are the commands to set up, compile, elaborate, and simulate the provided 1-bit adder VHDL example with its test bench. Note that the file named 1bit_adder.vhd starts with the digit '1' and not the letter 'l'.

```
$> source /cad/rc_scripts/rc.vhdl
```

{If not previously created and files copied:}

```
$> mkdir VHDL
$> cd VHDL
$> mkdir worklib
$> cp /cadhome/cadtest/VHDL/1bit_adder.vhd .
$> cp /cadhome/cadtest/VHDL/test_bench.vhd .
$> cp /cadhome/cadtest/VHDL/hdl.var .
$> cp /cadhome/cadtest/VHDL/cds.lib .
```

{Compile:}

```
$> ncvhdl –messages –v93 1bit_adder.vhd
$> ncvhdl –messages –v93 test_bench.vhd
```

{Elaborate:}

```
$> ncelab –messages –v93 worklib.ONE_BIT_ADDER:ONE_BIT_ADDER_RTL
$> ncelab –messages –v93 worklib.TESTER:TESTER_ARCH
```

{Simulate:}

```
$> ncsim -gui TESTER
```

{At this point a new *Cadence Affirma NC VHDL* window will open; specify remaining actions there.}

Select → Signals (pick all signals in *Cadence Affirma NC VHDL* window)

(invoke DAI Signalscan and open the waveform window)

(run the simulation)

ZmOutXFull (compress the X axis scale in the *DAI Signalscan Waveform:1* window to display all)

References

[1] Sharad Kumar, "*Quickstart Guide to Using Cadence AFFIRMA NCVHDL*", Michigan State University

[2] Cadence Openbook Manual, Cadence

[3] Affirma NC VHDL Simulator Help

Chapter 10: Bipolar Current Mode Logic

SiGe technology offers some of the speed of III-V devices (figure 10.1) at processing costs and yields of silicon. IBM has developed the capability to produce both 0.5 µm CMOS and 0.5 µm (50 GHz f_T) HBT (heterojunction bipolar transistors) devices in one process (which you the student have access to). Based on the f_T characteristics depicted in figure 10.1, one can observe the tremendous speed advantage of the SiGe Bipolar transistor.

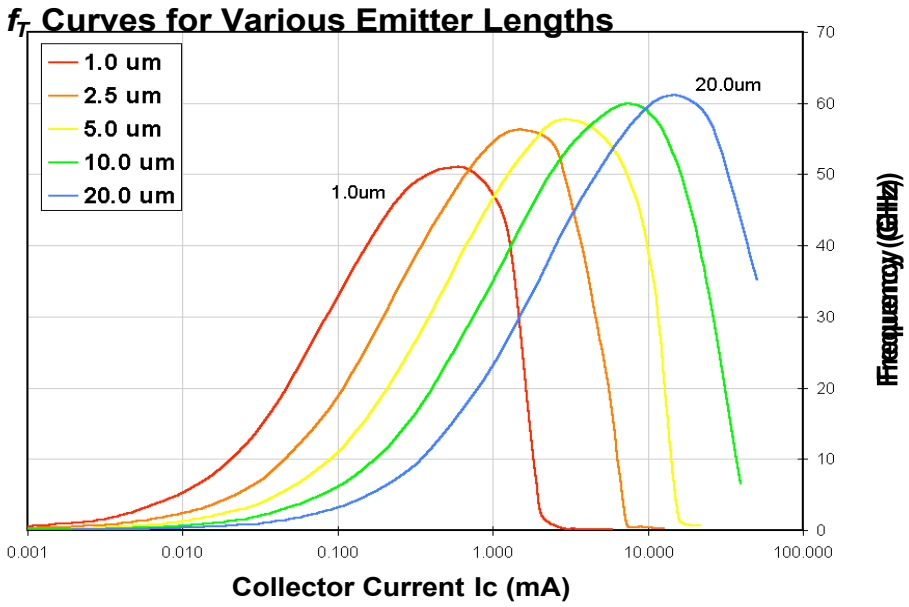


Figure 10.1: IBM SiGe f_T Plots

Bipolar transistors achieve their speed through the use of a very thin base region. The speed of the collector current depends on how long it takes for charge carriers to travel through the base. By introducing Ge into the base of a Si BJT, a smaller base bandgap is created which increases electron injection from the emitter and decreases the base transit time. The decrease in transit time results in a higher f_T and higher β. A higher β permits the base doping level to be raised, lowering the base resistance.

High speed in Current Mode Logic (CML) is achieved by operating all transistors out of saturation and keeping the logic signal swings relatively small. CML differential logic uses a pair of bipolar transistors as a current switch. By "steering" this current through different paths through differential logic control, current flow remains constant (figure 10.2). By examining the characteristics of the voltage levels of differential logic, over 99% of current will go down one branch if the voltage difference is 0.2 V. Using this type of logic produces low switching noise, a desirable feature when operating at high-speed. Another advantage of differential CML logic is that the complement signal is always available, so extra circuitry to create a signal complement is not necessary.

Figure 10.3 depicts a CML XOR implementation in 7 transistors and 3 resistors. Signal A and its complement come in on level 1 (0 and –0.25 V) and Signal B and its complement comes on level 2 (-0.95 and -1.2 V). The difference between levels is slightly more than one VBE (0.85 V). You may use 0.95 V as the difference to calculate new levels, therefore level 3 would have the signal and its complement be -1.9 V and -2.15 V. The tail resistor at the bottom of the current steering tree is connected to a reference current mirror that fixes the current through the circuit at 0.7 mA.

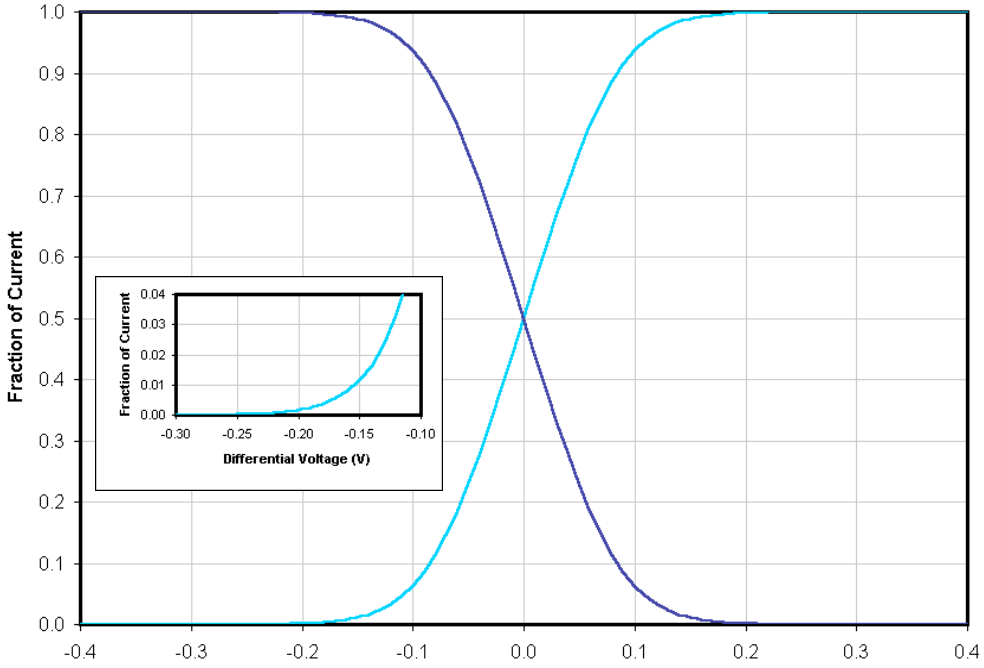


Figure 10.2: Differential Voltage vs. Fraction of Current

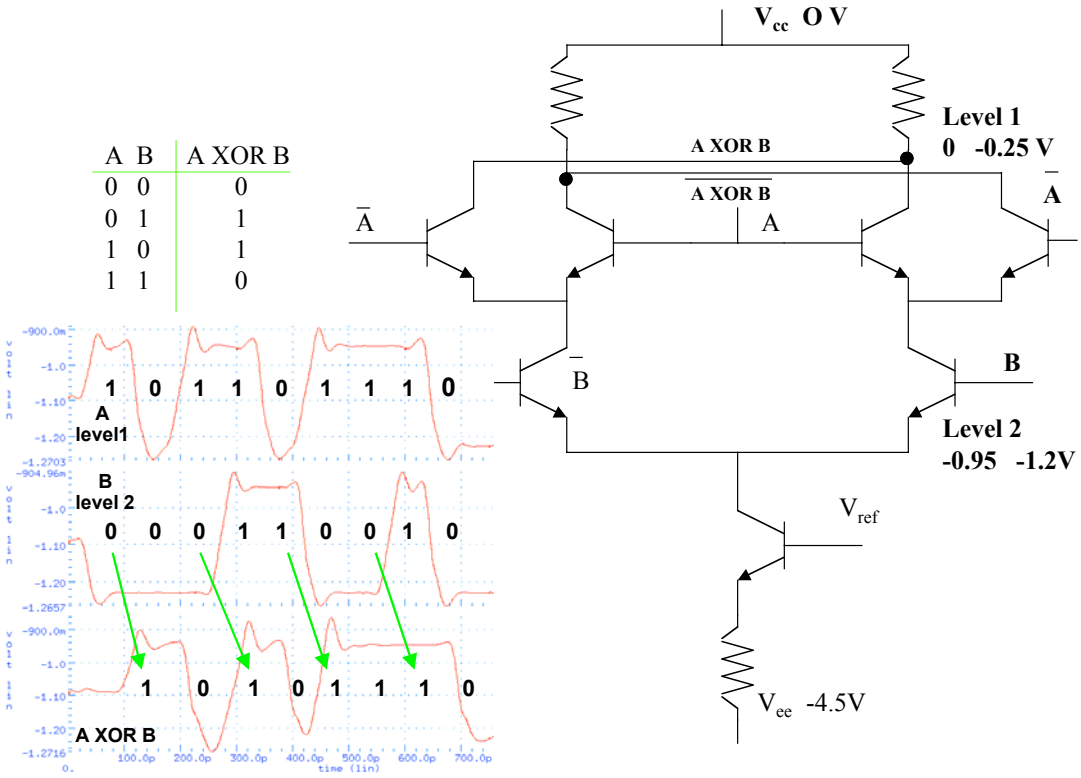


Figure 10.3: Full Differential Current Mode Logic XOR Cell

Current flows through only one path at a time. The voltage swing to turn on or off the opposite members of a given differential pair depends on what level the pair sits in the tree. This prevents saturation of any of the transistors, which slows down the switching of the transistor due to charge storage in this state. To avoid saturation, each transistor needs to have a bias or centering offset voltage that is different for each level. This requires a level translator circuit when the same driver drives multiple input levels.

10.1 Sample Bipolar OR Circuit

In the VLSI_CLASS Bipolar_OR_test, examine the various parts of the test circuit.

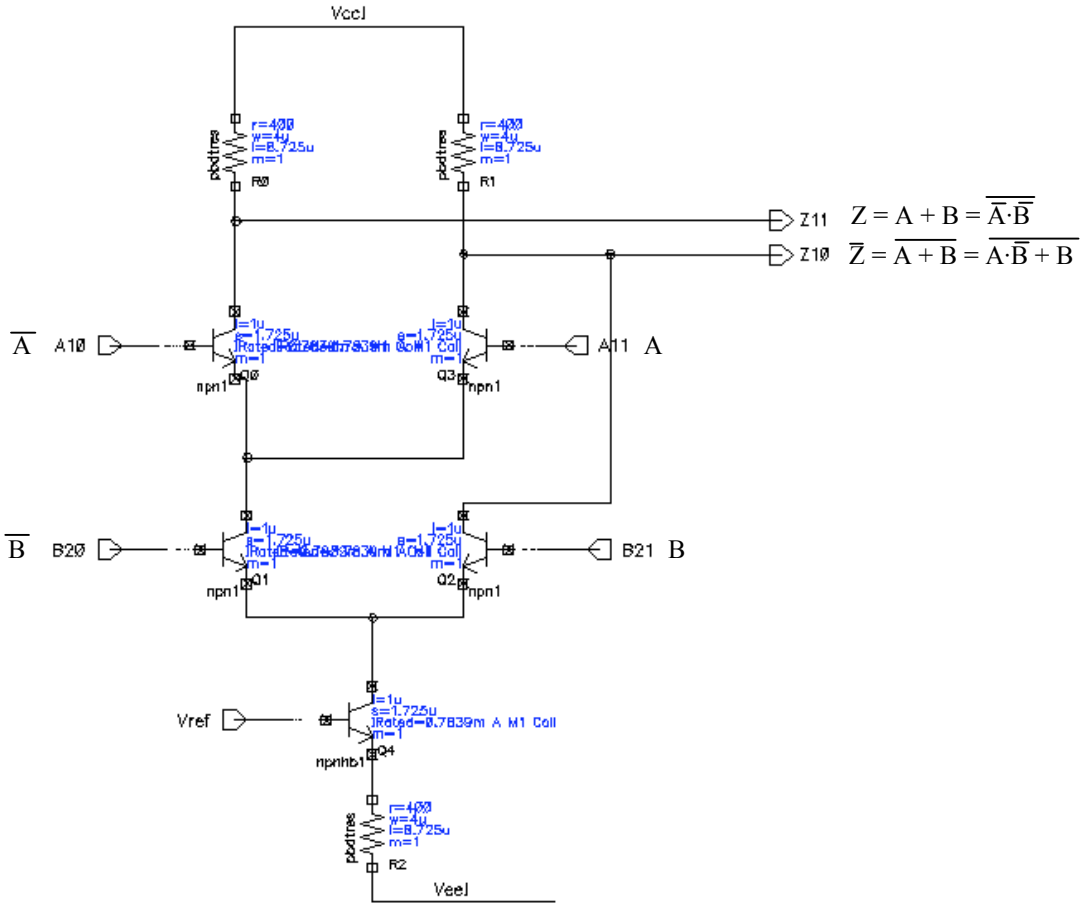


Figure 10.4: OR Schematic

Based on figures 10.1 and 10.2, we wish to have the max speed in our current tree. A 0.7 mA current flowing though a 400 ohm resistor will produce a voltage drop of 280 mV, very close to our desired 250 mV. Thus the level 1 (Z11, Z10) outputs will either be 0 (high) or –250 mV (low). Notice that the only way Z11 will be low is if A10 and B20 are high. Examine the 4 possibilities of A and B and convince yourself that this current tree performs the OR function.

The Vref is generated by a Widlar current source, which sets a DC reference current so that the same constant current flows in all trees (0.7 mA). A good exercise is to put DC current sources with 0 volts in each current path to measure the current in each path. (For reference, the V_{BE} for the SiGe bipolar transistor is 0.85 V. A slow npn1hb (high breakdown) transistor is used here since it has a higher current capability but never needs to switch. This transistor is often used in high power analog applications, the original purpose of the SiGe line.

Examine the bit pattern generators and notice how they are constructed. They use a series of complementary voltage pulses to generate the input pattern. After 600 ps, the pattern repeats. By changing the timescale, one can produce faster/slower signals to stress the circuit. The purpose of the buffers is to “dirty” the signals to make them more realistic, since we do not have nice square waves when dealing with high-speed signals. The purpose of the buffer on the output is to put a load on the output signal.

Since are working with descending level current trees, it is OK to send a level 1, 2, or 3 signal to a level 1 input, but a level 1 output cannot be sent to a level 2 or 3 input (they are lower in the tree). You will have to remember this when making your adder.

When designing for very high-speed operation it is good to make sure that all voltage drops in the current tree have symmetry with respect to the other side of the current tree. In situations where this is not the case add diodes to compensate for the asymmetry. This is important since it reduces the magnitude of voltage changes in the circuit and diminishes the significance of capacitance effects.

10.2 Layout of Bipolar

Most of the layout rules you learned doing the CMOS adder still apply. Since we are dealing with larger currents, we have to pay attention to the wire widths and how much current is flowing. Here are some useful formulas relating current in mA to wire widths in μ:

Metal 1 current = 0.74·(w - 0.27)
Metal 2 - 4 current = 1.1·(w - 0.13)
Last Metal current = 1.1·(w - 0.13)

Thus if you are going to route where there is 0.7 mA flowing, use 1.6 μ if using metal 1 and 0.9 μ for metals 2 - 4. Otherwise use the minimum pitch wire for low current areas, such as the base connections.

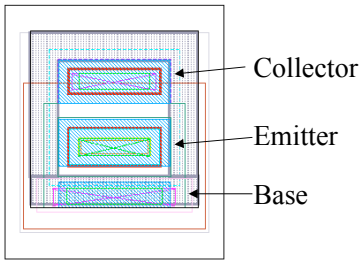


Figure 10.5: NPN SiGe Transistor

You should have the required tools to now layout the other cells and wire your bipolar adder together.

Chapter 11: Standard Cells

11.1 Creating a standard cell introduction

The creation of integrated circuits is an extremely complicated undertaking. Over the years as the level of complexities in the circuits gradually increased new methods of design were invented. Designers realized that there was a lot of wasted time when certain logical functional units were recreated each time a new design was created. This process was similar to re-inventing the wheel. The use of standard cell and macro cells developed as a means to increase productivity and reduce wasted time.

11.2 Standard Cell Properties

Standard cells are cells that have certain properties, standardized such that they can be easily integrated with other standard cells. A standard cell normally has a pre-defined width and the "Vcc" and "Vee" metal layers in the cell are of a predefined width. The spacing between Vcc and Vee is also pre-defined. These cells when created using a bipolar CML (Current Mode Logic) digital process contain an extra ubiquitous last metal layer, the reference voltage. In consequence a bipolar standard cell has slightly different specifications than a CMOS standard cell. We will focus on creating bipolar CML standard cells.

Standard cell specifications:

Total Width	= 69 μ
Vcc last metal width	= 30 μ
Ground last metal width	= 30 μ
Spacing between last metals	= 3 μ

The width specifications of various layers in the cell aren't the sole characteristics of standard cells. The manner in which the routing is done and the placement of the inputs and outputs to the cell must follow certain guidelines. These guidelines may be broken when the design necessitates such a measure; here one must use common sense. In a standard cell the metal1 is generally used for vertical routing and metal2 for horizontal routing. The rules allow for pass through connections to be made through cells without having to contour the cell. They also, when properly implemented, allow for more dense cells and simplify the task of the router. A standard cell library may be further optimized, by making the inputs and output of the standard cells match up. Such a modification allows for the cells to be easily placed in series. The inputs and outputs to a cell may also be placed at the top and bottom of the cell in order to allow for simplified automated routing.

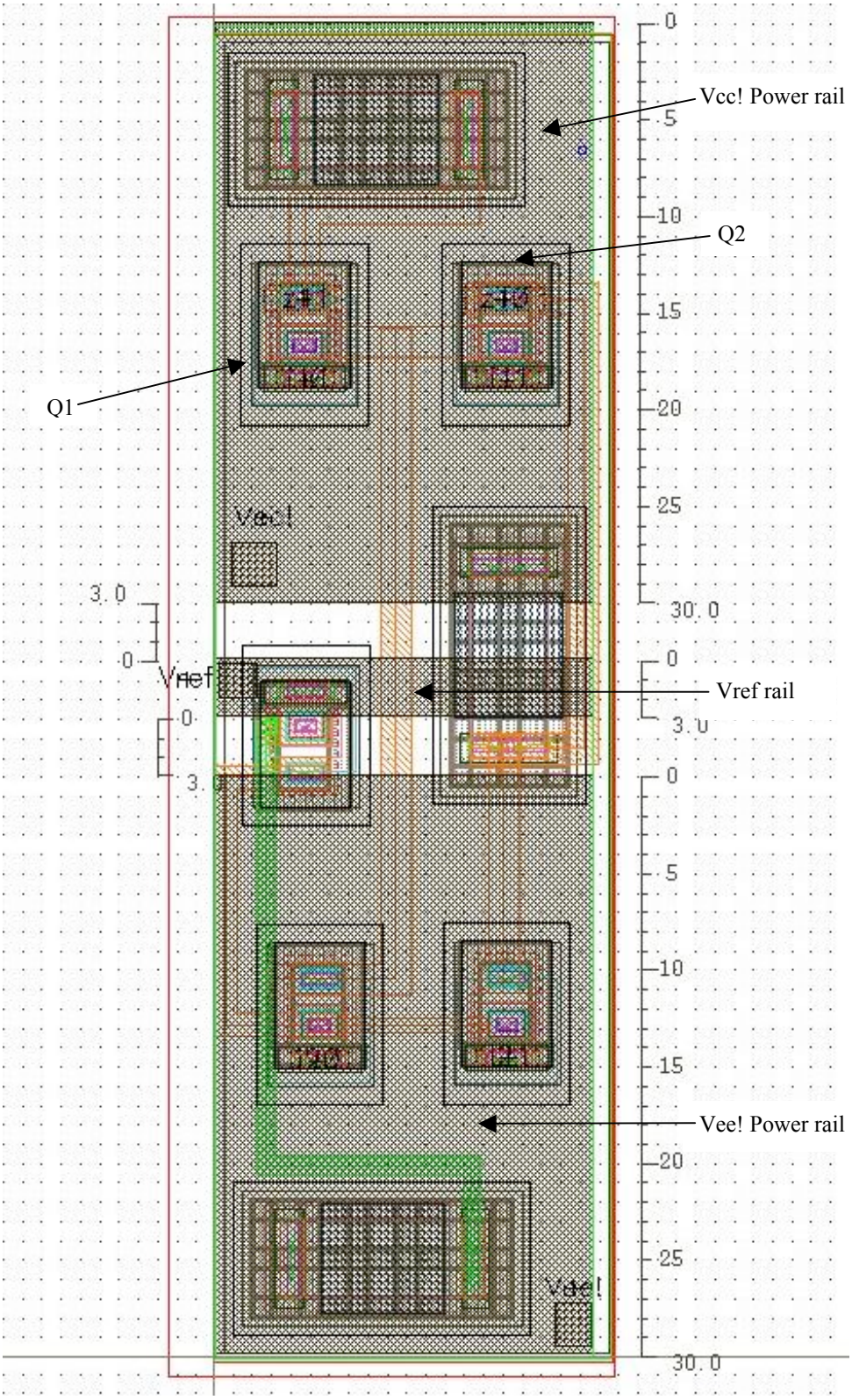


Figure 11.1: 2-input AND Standard Cell

11.3 Creating a Standard Cell

1. All bipolar cells have a Vcc, Vee, and Vref rail. Creating these rails first is a good place to start laying out the cell. Use the **Rectangle** command by pressing "**r**".
2. The next step involves placing instances of the transistors into the layout window. The transistors come as pcells (parameterized cells), therefore you may obtain different variants of the transistor by changing certain parameters in the instance window. Create the transistor instances required for an XOR. Use the **sigehp** library to create the instances. Note that the transistor at the bottom of the current tree is a high breakdown transistor (npnhb1). Use instances pbdtres to create the resistors and npn1 for the transistors. Note: The parameters in the layout should be identical to the parameters in the schematic.
3. Place the appropriate devices in the appropriate locations keeping their respective connections.
4. Make the appropriate connections between devices, limiting yourself to the use of metal1 and metal2.
5. Create the **Vcc**! And **Vee**! power rails and the **Vref** line and make the appropriate connections.
6. Create the cells pins. The pins should have identical names as the ones used in your schematic.

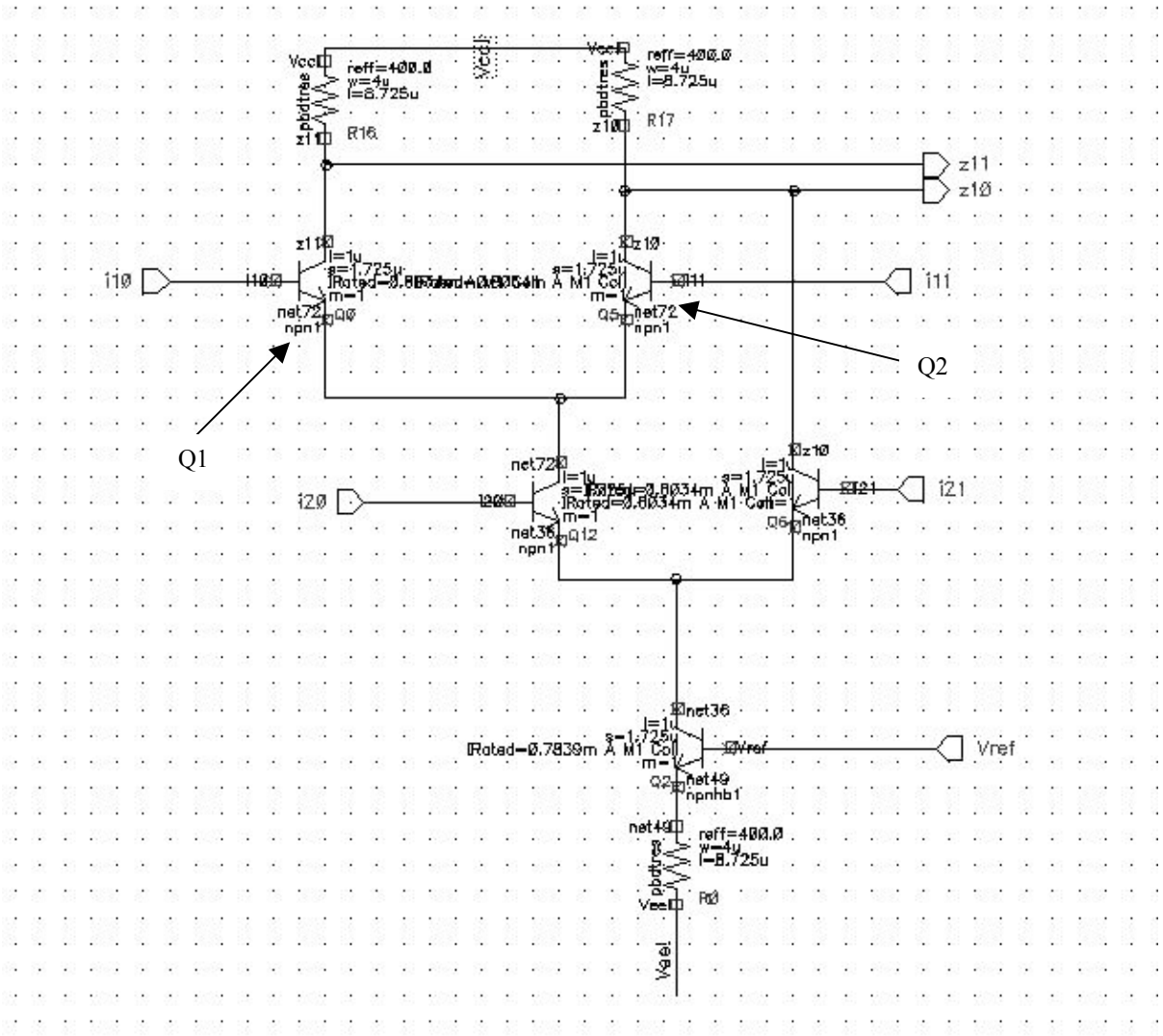


Figure 11.2: 2-input AND CML Schematic

7. Once the cell is created place a **prbound** drawing and an **instance** (labeled as instance dg in the layout view) rectangle around the cell. These enable the tool to know where cell begins and ends.
8. Perform a **DRC** (**Design Rule Check**) to ensure that you have not broken any design rules.
9. Perform a **LVS** schematic check to ensure that your layout is an accurate rendition of the schematic.

11.4 Issues Using Standard Cells

Standard cells impose certain limitations on the flexibility of the design since the cell has various parameters standardized. It limits the type of logical functions that may be implemented since these need to fit within the standard cell. A voltage controlled oscillator would not be a good candidate to be implemented within a standard cell. The size of the last metal lines also imposes limitations upon the design. The last metal needs to be wide enough to sustain the current drawn from all the cells connected to that last metal line. The rule which specifies the current a line can withstand is current = 1.1·(width - 0.13), for DC current on the last metal lines. A designer must also take note that the probes used to power a chip can normally sustain 0.5 A per pad. Voltage droop may arise on the line if the restraints are not kept in mind.

11.5 Further Examples

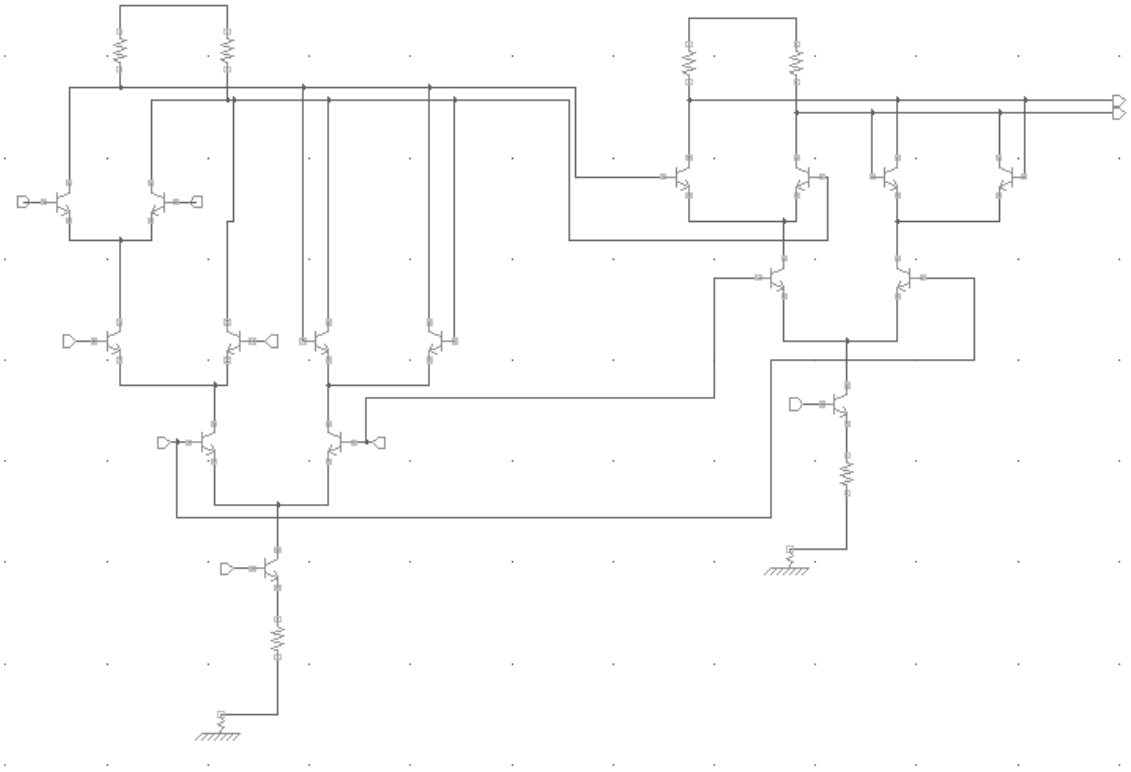


Figure 11.3: Schematic Master-Slave Latch With 2-input AND

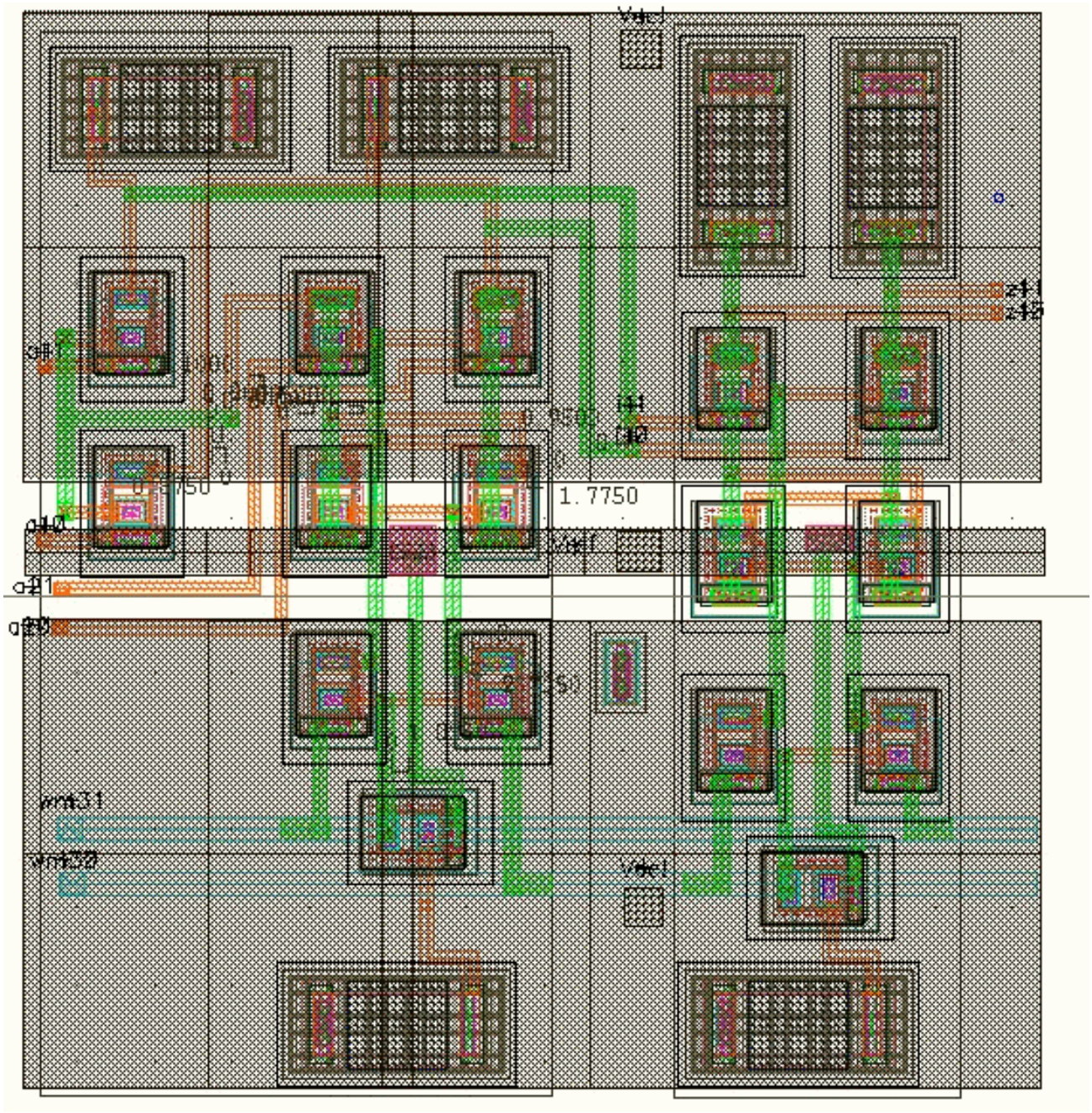


Figure 11.4: Layout Master-Slave Latch With 2-input AND

Chapter 12: Routing and Placement

12.1 The Routing and Placement Tools

The design process was greatly simplified when producing updated circuits due to the advent of standard cells and macro cell. Yet while producing extremely complex circuitry issues such as placement and routing which used to be relatively simple became very time consuming. Optimizing the wire lengths, through the placement of devices and the paths taken by the wires, for thousands of wires created an impediment to having a quick time to market. As a result placement and routing tools were created to increase productivity. The placement and routing tools used algorithms that cannot equal the capabilities of a good engineer, yet they are very effective for average ASICs that don't require maximum optimizations to attain their speeds.

The placement and routing tools are used to place and route while adhering to the design rules specified by each respective process. The design rules restrict the size and type of geometries usable by the process layers. The initial step in setting up a placement and routing tool is to configure its design rules file according to the process' design rules.

12.2 CCT Router

The CCT router is a program that creates the wire inter-connections between nodes in the circuit. The CCT router allows the design engineer to create a quick layout of the circuit and use the tool to perform some of the more tedious optimizations. A great advantage of the tool is its capability to perform differential routing. The CCT router was recently purchased by Cadence, and its functionality is incorporated within the IC Craftsman tool. The IC Craftsman router reads in the placement and connectivity information to create a model of the circuit. The instances present in the circuit are represented as images, which may not be manipulated further. The router displays the recognized vias and wires as such. The circuit within the CCT router will appear different from the layout view within Virtuoso, but when converted back to Virtuoso it will revert to its original format.

12.3 Converting a Virtuoso Layout to the IC Craftsman tool

In order to use the IC Craftsman routing tool one must first convert the design into a format readable by the tool. The Design Framework has a conversion utility to convert the design to the IC Craftsman format. The converter creates seven files, which define a different aspect of the design.

IC Craftsman files

.dsn	Is the main declaration file, it refers to the other files and declares the instances used.
.img	Defines the instances and the via arrays. This file allows the ICC tool to provide a vague geometrically accurate representation of the instances converted.
.clr	Defines the colormap, so that layers remain the same color.
.net	Defines the I/O nets.
.pla	Contains placement information for the images (instances).
.str	Provides a rules template so that the ICC tool is aware of the process' design rules.
.wir	Describes the wiring present in the circuit.

12.4 Conversion steps

1. Open the schematic view of the design you wish to convert. (Note: All the instances in the design must have an accurate layout and the pinouts between layout, schematic and symbol view must correspond.)
2. Open a new *Virtuoso XL* window by click on **Tools → Design Synthesis → Layout XL**. Virtuoso XL is a placement tool. The tool allows us to partially automate the process of placing instances in the layout with their corresponding pinouts and connectivity information.
3. At Virtuoso XL startup, option click on **Create New** followed by **OK.**
4. When the *Create New File* form appears, click **OK.**
5. The tool will open a Virtuoso XL layout window. Click on menu item **Design → Gen from Source…**.

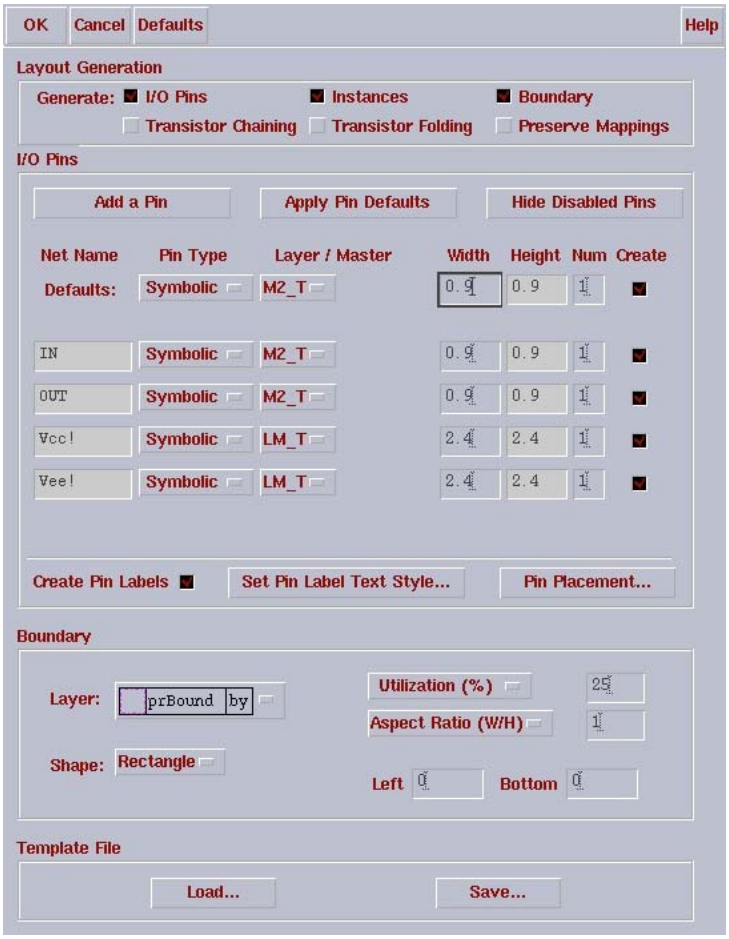


Figure 12.1: Layout Generation Options Form

6. The *Layout Generation Options* form will open.
   - Change the **Pin Type Defaults:** to **Symbolic** and the **Layer / Master Defaults:** to **M2_T** (it has the best R/C characteristics).
   - Press **Apply Pin Defaults**.
   - Change the '**Vcc!**' And '**Vee!**' pins **Layer / Master** to **LM_T**.
   - Set **Utilization (%)** and **Aspect Ratio (W/H)**. (Note: The utilization % refers to the level compaction of the placement; 70% is considered very good.)
7. Click **OK.**

The tool should generate a layout of your schematic based on the layouts within your instances. The placement tool does not generate a very good placement, but you may manually modify the placement of the components within the *Virtuoso XL* window to optimize the placement. Select the component with the left mouse button and press the **M** key to move it. Keep in mind what is connected to what such that your placement does not make the routing much more difficult.

8. Click on the menu item **Route → Export to Router**. The *Export to Router* form will appear. Set the **Export Layout Cellview:** appropriately. Set the **Export Netlist From:** item to **Schematic Cellview** and enter the **Library** and **Cell** of your design's schematic view. You may use the two **Browse…** buttons to identify both the layout cellview and netlist.
9. Click on **Use Rules File** and set the field to point to ~cadtest/iccraftrules.rul
10. Unselect the **Start Router with options:** bullet, set the **Export to Directory:** to where you would like the design file to be created and press **OK**. This should export your design in an IC Craftsman format.

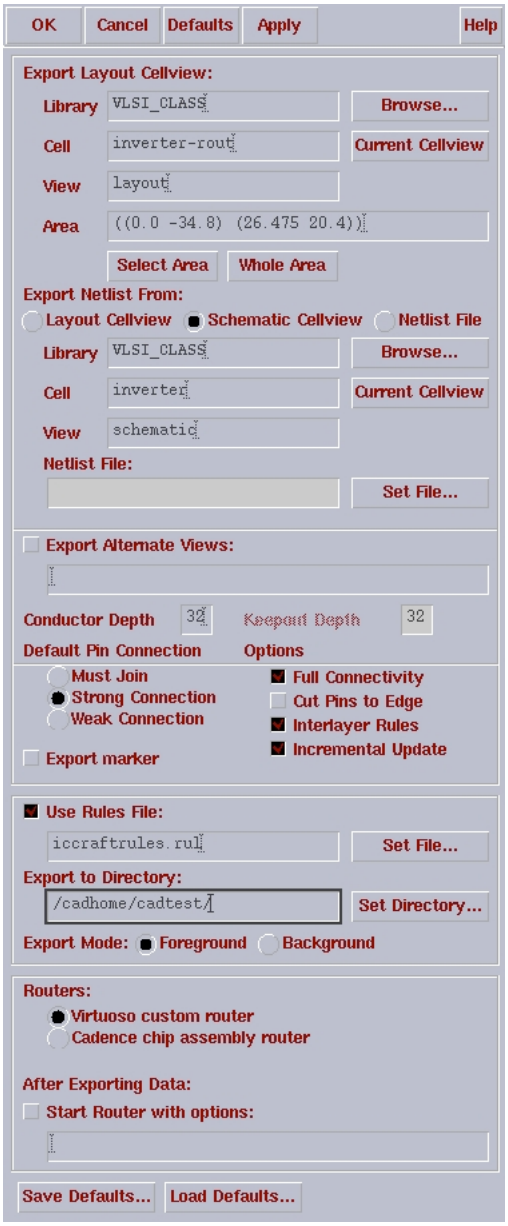


Figure 12.2: Export to IC Craftsman Form

12.5 Running the Integrated Circuit Craftsman Tool

The layout view may contain either placement information in conjunction with the I/O pin information or rough routing in conjunction with placement information. The placement information with some routing information is already done in a layout, either manually or with another automated tool. The IC Craftsman tool, when importing the design files, creates an equivalent version of the layout. The IC Craftsman version of the layout contains information on where the instances are placed, and where the routable layers are. These are shown in the IC Craftsman tools window with similar colors as those found in the Virtuoso tool.

12.6 Opening a Converted Virtuoso Layout

1. Go to the directory where your exported design files are located and type **ic_craft &** at the prompt to run the IC Craftsman tool. The tool will open by default in the graphical user interface based startup window. The window provides you with the option of opening a design file in conjunction with various other files. These files allow a user to automate various aspects of the placement and routing process by incorporating many of the actions, usually performed using the GUI based menus in the IC Craftsman

tool, in a file. In order to use the IC Craftsman tool you need either a design file or a session file. The design file that was created when exporting the layout from the Virtuoso tool (.dsn suffix) should be selected with the **Browse…** button for **Design / Session File:** file.

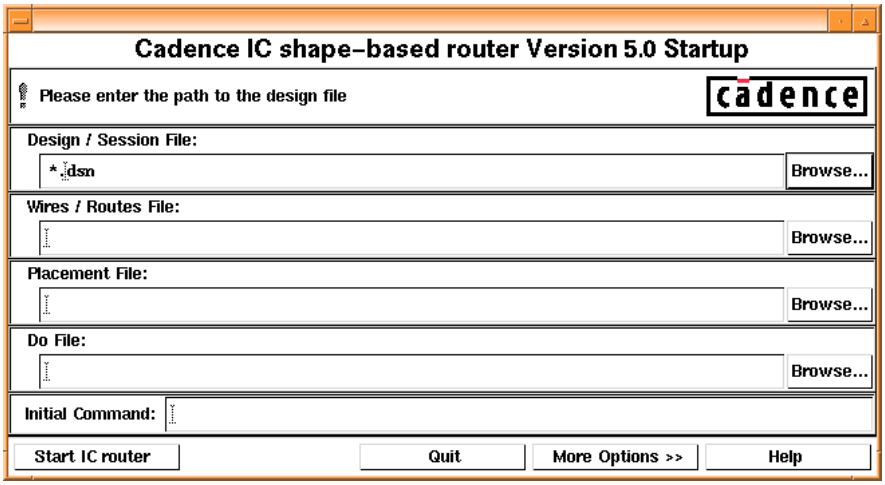


Figure 12.3: IC Craftsman Startup Menu

2. **Click** on **Start IC router**. This will open the IC Craftsman tool with your IC Craftsman version of your layout. Your layout will be placed as it was by the Virtuoso XL tool and it will have solid white lines representing the parts of the design that need to be connected by the routing tool.

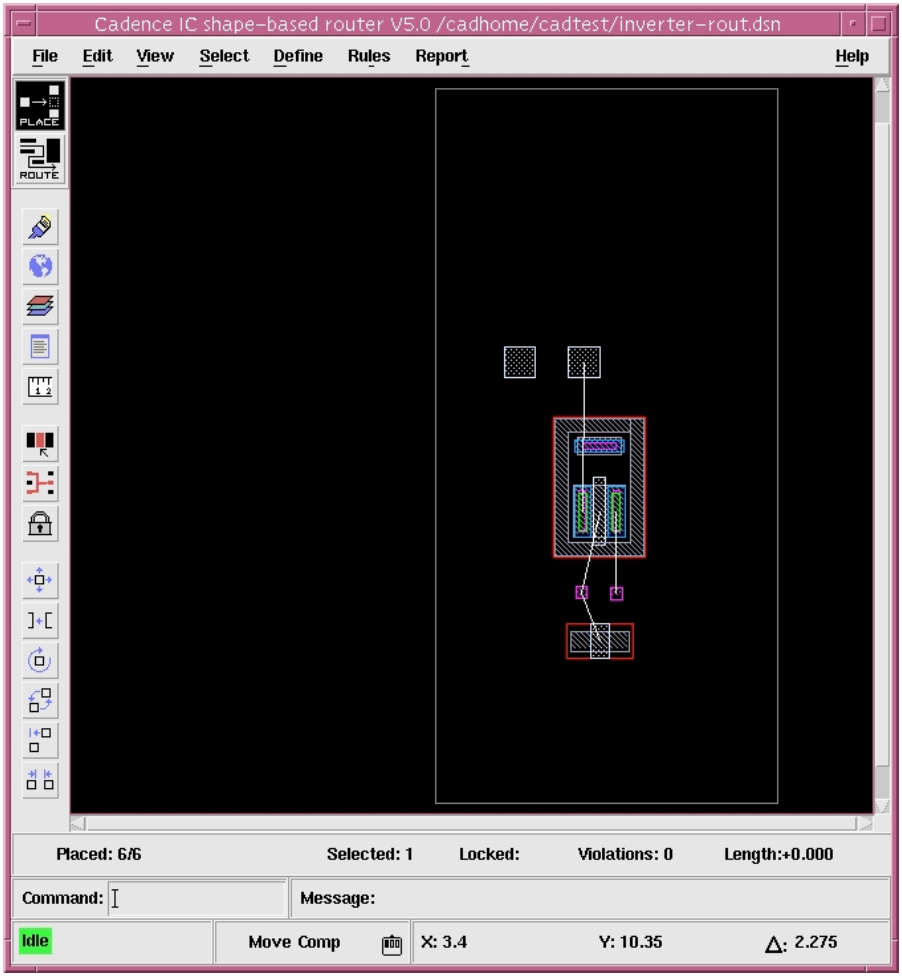


Figure 12.4: IC Craftsman Router Tool Window

12.7 Routing an Imported Design

1. Defining the differential wire pairs is the first step before setting up the circuit for routing. In order to define the wire pairs, click on the menu item **Define → Net Pair → Define/Forget By List…**.

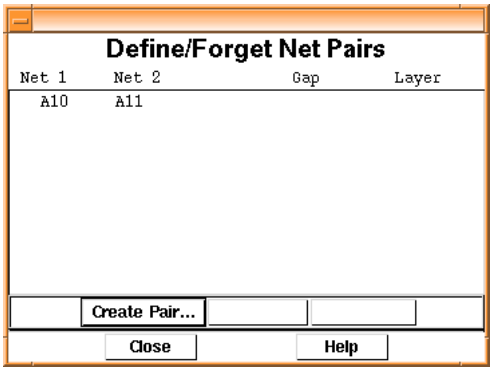


Figure 12.5: Selected Net Pairs List

- An interactive *Define/Forget Net Pairs* window will open. This window allows you to define which nets are to be routed as a pair as well as define at what distance, if any, this pairs must track each other.
- **Click** on **Create Pair…**.
- A second window will open. This window contains a list of the nets that the IC Craftsman tool recognizes. These nets were obtained form the translation of the connectivity information.
- They are two lists **Net 1** and **Net 2** click on a net pair using each respective window. This selects which two nets are to be treated as a pair.
- You may set the tracking distance by **clicking** on the square to the left of **Set Gap:**. This will enable the gap setting capabilities.
- Set the tracking gap in microns or use "–1" if you don't want any tracking restrictions. One must remember the more rules you impose on the routing, such as tracking, the longer the router will take to route the circuit. A condition where routing is impossible may also arise as a result of overly restrictive rules.
- Once the net pair is selected and configured **click** on **Apply** and repeat the operation for all the nets that need differential routing. When done, **click** on **Close** in the *Define/Forget Net Pairs* form.

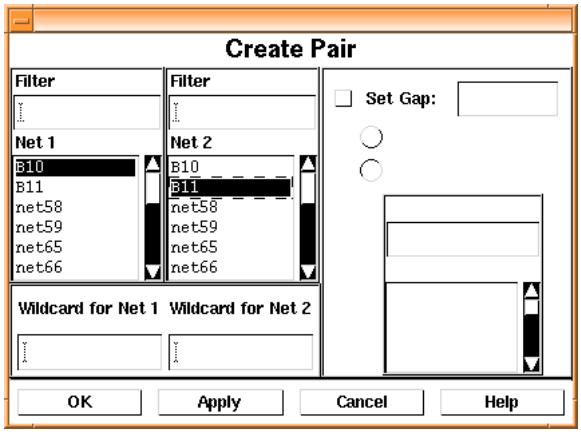


Figure 12.6: Net Pair Selection Form

2. The next step is to initiate the routing of the layout. Routing is not a one step process. It may take several passes and one might need to perform what is known as a clean pass. The route command runs

basic autorouting passes. With each routing pass, the autorouter tries to route connections that are not yet routed and reroute connections that are involved in conflicts or are close to wires involved in conflicts. If you did not select any connections, the autorouter tries to route all connections defined in the network, except those that are fixed or protected. If you select any connections, the autorouter tries to route only those connections that you selected. The clean command rips up and reroutes every connection, removes unnecessary vias and bends, and alters the routing problem by making new or different routing channels available for the next series of route passes. You will see a noticeable improvement in the routing quality after the clean passes.

3. **Click** on menu item **Autoroute → Route…** and select the number of **Passes:** you want the router to undertake. Generally the more passes the better, yet there may be situations where a change in the design or the rules is needed instead.
4. When the passes are complete run the clean command with the menu item **Autoroute → Clean…**.
5. Repeat steps 3 and 4 until results are satisfactory.

Certain designs, due to the placement of the devices and the limited space provided for routing as a result of how close certain objects are allowed to be, may be inherently unroutable. In such a case the designer needs to go back to the Virtuoso XL tool and modify the placement accordingly or otherwise use the placement capabilities incorporated within the routing tool to solve such problems. In certain cases, if you are routing standard cells, the closeness of certain pinouts in the standard cell may be at fault and the standard cells themselves would need to be reworked.

Appendix A: Useful Shortcut Keys

Right click box to zoom in, small box, faster zoom

o	zoom out 2x
O	zoom out 5x
z	zoom in 2x
Z	zoom in 5x
s	stretch
c	copy
m	move
g	toggle gravity
cntrl d	deselect items
del	delete
p	pin
P	convert path to polygon
r	rectangle
R	redraw window
w	wire
W	bus
i	instance
Q	properties
E	editor

F3 changes characteristic like orthogonal to any angle moving

Cadence On-line Documentation:

For IBM AMS Kit Design Rules - in the terminal window, type:

`cd /cad/cds/amsv2.4/doc`	
`ls`	(to see the list of PDF files in the directory)
`acroread filename &`	(where *`filename`* is the PDF file to viewed)

For Full Cadence documentation – in the terminal window, type:

`openbook`	(this opens the OpenBook main window) The default page will be for IC. To change to another manual such as LDV or ICC, select **Window → Set Preference…** and in the new dialog box, clicking on the Current Active Documentation Hierarchy choice will display other available manuals. Select another one and click **Apply**.