

# Custom Compiler Training

## - Foundation and Schematic

# 1

## 1. Custom Compiler Platform Overview

### Learning Objectives

In this lab, you will learn how to:

- Invoke and exit the Custom Compiler platform.
- Open a design.
- Navigate the GUI windows to view objects of interest.
- Use the Preference Manager GUI.
- Navigate the viewport using the Bookmarks Manager.
- Use the Display Resources Editor GUI.
- Use command -help to get additional Custom Compiler command information.

In this lab, you will learn how to:

- Invoke and exit the Custom Compiler platform.
- Open a design.
- Navigate the GUI windows to view objects of interest.
- Use the Preference Manager GUI.
- Navigate the viewport using the Bookmarks Manager.
- Use the Display Resources Editor GUI.



**Lab Duration:**  
30 minutes





# Before You Start

Make sure you have the following:

| Item   | Description  | Location   |
|--|--|--|
| Custom Compiler Platform<br>O-2018.09                            | Software   | Installed on the server                          |
| Custom<br>Compiler_Overview_Lab1:<br><br>DemoVco<br><br>lib.defs | Lab data folder:<br><br>OpenAccess design<br>library | The directory where you<br>unpacked the lab data |
| PDK  | reference40nm Technology<br>library                  |  |

## Answers & Solutions

Each lab contains answers to all questions and results or solutions.

*You are encouraged to verify your results by checking the **Answers/Solutions** section at the end of each lab.*

# Instructions

Follow the instructions in the lab to explore Custom Compiler platform basic features. Please see the “Console and Design Editor” section of the Foundation command reference manual for the commands used in this lab.

## Task 1. Starting the Custom Compiler Platform

---

1. In the UNIX window, change the directory to Custom Compiler\_Overview\_Lab1.

```
Unix% cd Custom_Compiler_Overview_Lab1
```

2. Start the Custom Compiler platform from the UNIX prompt using the shell command **custom\_compiler**.

```
Unix% custom_compiler &
```

Launching the Custom Compiler platform creates the command file, *custom\_compiler.tcl* and the log file, *custom\_compiler.log* in *synopsys\_custom* directory. The Custom Compiler Console can be opened on launch if the **giShowFrameTabs** preference is false or by clicking the Console icon on the Home page.


**Note:** All command entry and logging is controlled by the Console window. Additional features can be accessed through the menu bar entries.

## Task 2. Using the Preference Manager

---

The Preference Manager provides a set of configurable options to change the application behavior.

In this task, you will invoke the **General Options** preferences window and explore its contents.

1. Open the Custom Compiler Console.
2. Choose **Options > General** from the Custom Compiler Home Page. The **General Options** window displays the general options applicable for all the design editors such as Layout Editor, Schematic Editor, and Symbol Editor.
3. Observe that in the **General Options** window, the **Binding Set** is **Custom Compiler**, the default binding set defined for the application.
4. Click the button  to the right of the **Defaults** and browse through the contents of the menu. Use this menu to save the preferences to the predefined locations.

**Note:** For each design editor, you can set specific preferences by calling the respective **Design Options** window. Call the specific **Design Options** window using the menu **Options > Design** from the design editors.

**Question 1.** Save the preferences from the **General Options** to a file *test.xml*.

.....

5. Cancel the **General Options** window.

### **Task 3. Creating Custom Bindings**

---

Individual commands can be bound to the keyboard key sequences to accelerate the design process. The Custom Compiler platform provides options to create or modify the existing binding set.

In this task, you will create a binding set and add a new binding to the existing binding set.

1. Create a new binding set called *MyBindings* using the **gi::createBindingSet** command.

```
gi::createBindingSet "MyBindings"
```

2. Choose **Options > General** to open the **General Options** window. Click **Binding Set and** you will see that *MyBindings* has been added to the list.
3. You can add a new binding using the **gi::createBinding** command. The binding will be created and added to the current active binding set.

In the next task (Task 4), you will define a bindkey for redrawing the layout, which will be used in this lab.

**Note:** The binding set is active only for the current session. This can be sourced by default by saving the bindings to *bindings.xml*, at any of the standard startup locations.

### **Task 4. Learning OA, program and Tcl versions**

---

Sometimes scripts may have dependency on the program, Tcl or Open Access database version. In this task, you'll learn commands to obtain the required information.

1. To find TCL version type the following in the Custom Compiler Console:

```
>set tcl_version
```

```
8.6
>info patchlevel
8.6.3
```

2. To find OA version, one should rely on OA Tcl commands

```
oa::getBuildName [oa::BuildInfoFind oaBase]
22.50.064
```

3. To obtain the program version type the following

```
db::getattr version -of [db::getProcessInfo]
O-2018.09
```

4. Another useful command to test the program version is db::checkVersion

```
db::checkVersion -atLeast 2017.12
1
db::checkVersion -atLeast 2019.03
0
```

## Task 5. Finding Commands

---

The Custom Compiler platform supports command listing and command line completion through the **Tab** key. In this task, you will find the command to create a binding and also create a bindkey using the command.

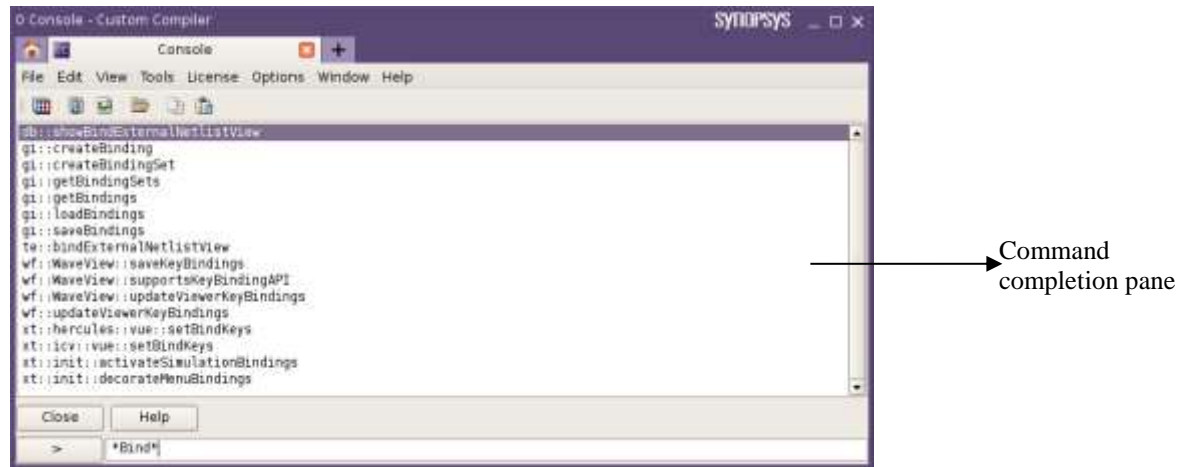
5. Try the following in the Custom Compiler Console:

```
>*Bind*<Shift-Tab>
```

**Note:** Observe that command completion pane is brought up after entering the preceding in the console.

6. Executing the preceding lists in the console all of the commands which contain “Bind”. If more than one possible match exists, the completion will stop at the point where the matches differ.

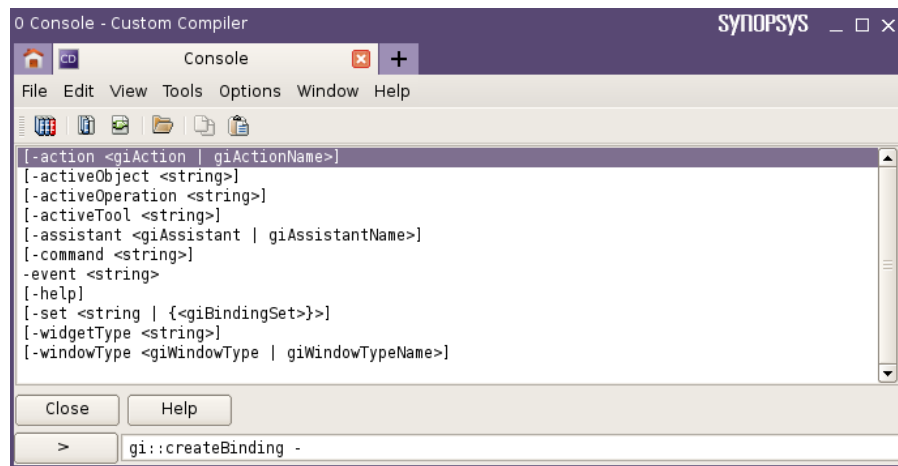




7. Use the **Up** or **Down** arrow keys to scroll through the list. Select and double-click or press the **Enter** key to select the command **gi::createBinding**.

**Note:** Observe that the command completion window closes after you select the command.

8. To find the syntax of the command, in the Custom Compiler Console, press the **Tab** key to bring up the command completion pane.
9. Use the **Up** or **Down** arrow keys to scroll through the list. Select **-help** and press the **Enter** key to see the syntax of the command.



**Note:** Another way to find the syntax of a command is to type the command name followed by `-help` in the console.

```
> [command name] -help
```

**Question 2.** Find the syntax of the command **dm::createLib**.

10. Type the following in the console.

```
> gi::createBinding -e<Tab>
```

**Note:** Observe that pressing the **Tab** key after “-e” autocompletes the option name **event**. Similarly, all option names can be autocompleted.


11. Complete the command and create the binding using the following as a reference.

```
> gi::createBinding -event "Alt-7" \  
-windowType "leLayout" -command \  
{de::redraw -window %w}
```

## Task 6. Opening the Design Editors

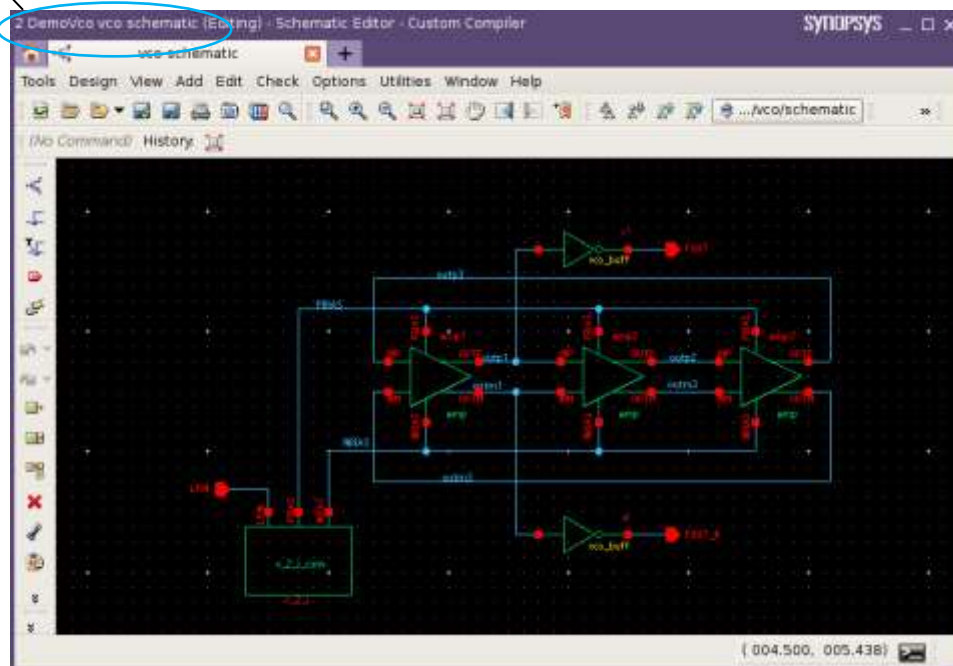
---

In this task you will open the previously created designs of different cellviews with their respective editors.


1. Load the *vco* schematic cellview from the *DemoVco* OpenAccess library:
  - a. Click **Open**  on the toolbar or choose **File > Open Design** from the console.
  - b. In the **Open Design** window, click *DemoVco* under **Libraries**. Selecting the library lists the cells in the library under **Cells**.
  - c. Enter ‘vc’ in the field above **Cells**. **Cells** is modified to list only cells that contain *vc*.
  - d. Select the *vco* cell. The available views of *vco* are listed in the **Views** column.
  - e. Select the *schematic* view from **Views** and click **Open** to open the cellview.

The cell is opened in the Custom Compiler Schematic Editor. Observe the placed cell in the Schematic Editor window. The *vco* cell is created by instantiating symbols. In addition, note that there is information displayed about cell on the upper-left corner of the window.

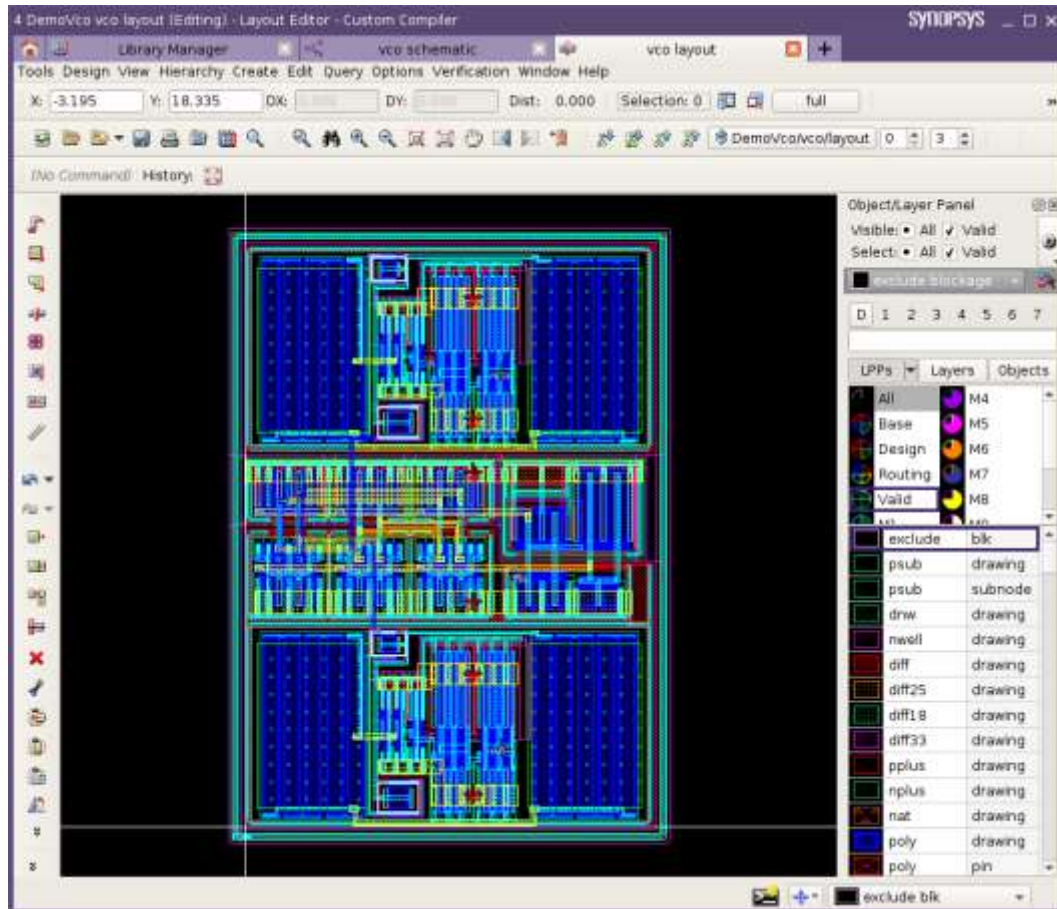
Library Name  
Cell Name  
View Name



**Note:** The Schematic Editor window has its own set of menu entries. Some of these entries are shared with the Main window; others are unique to the Schematic Editor.

2. Choose **Design > Close** to close the Schematic Editor window.
3. Open the *vco* layout cellview using the steps explained in Steps 1a through 1d. Instead of the *schematic* view, select the *layout* cellview. This opens the layout view of the *vco* cell in a new Layout Editor window.
4. By default, only the top level data will be drawn. Change the preference value from **Options > Design**, select **General tab**, and set **Hierarchy Levels: Stop** to 3.
  - a. Save the changes made in **Options> Design**, to the cellview scope, using **Defaults**  **>Make Defaults for > CellView**.
  - b. Click **OK** to close **Layout Design Options** window.

**Note:** When the cell is opened the next time, the cell data drawn will be up to hierarchy level 3.



**Note:** The Layout Editor window has its own set of menu entries. Most of the physical processing of the design can be done by commands from the menu.

## Task 7. Viewport Navigation in the Layout Editor GUI

1. Spend a minute to get familiar with the zoom and pan buttons in the Layout Editor window.

**Hint:** A short descriptive ‘Tool tip’ will pop up when a mouse pointer parks over a button.


Set Viewport, Show Magnified Design, Zoom In, Zoom Out, Fit to Selected, Fit to Edit, Pan, Previous/Next Viewport, Add Bookmark



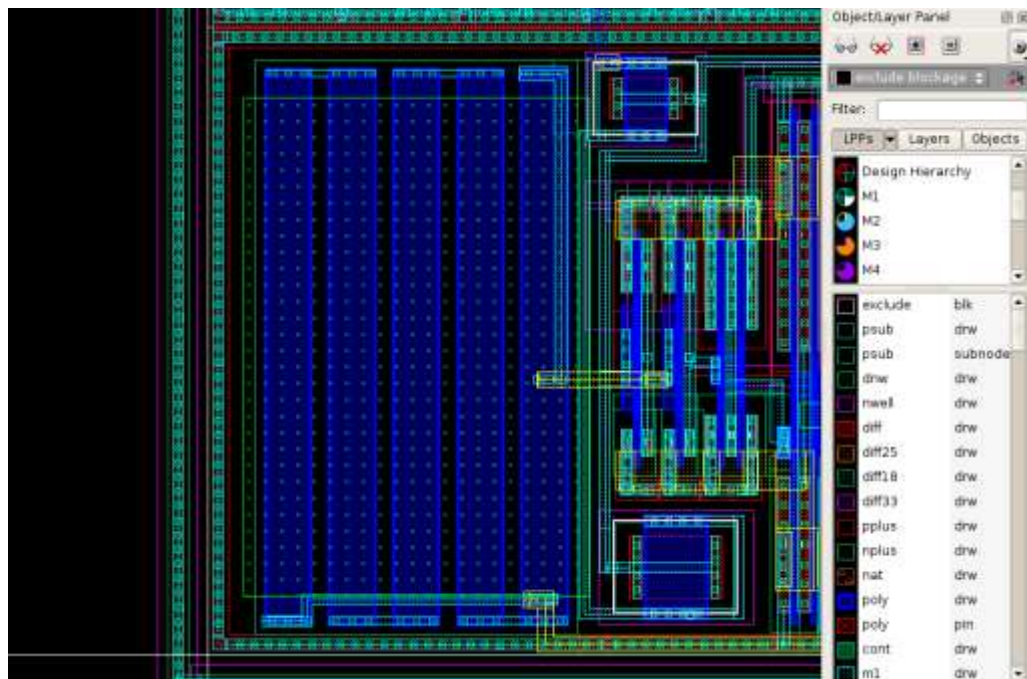
**Note:** Bind keys are also available when a design editor window is active.  
For example, in the Layout Editor window, **f** corresponds to zoom full.

**Note:** The Layout Editor window maintains a history of pan/zoom views. Views can also be saved. This can be achieved by using the Bookmarks Manager.  
This is applicable to the other design editor windows.

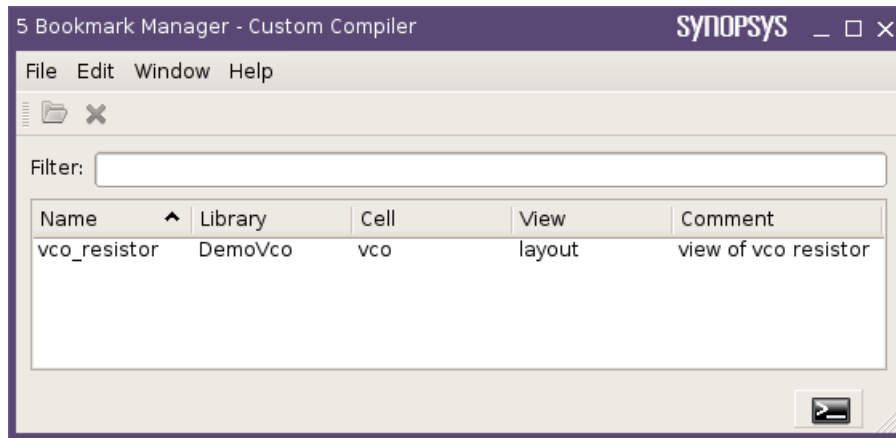
2. Make the Layout Editor window active.

**Note:** Interactively zoom in to the resistor at the left of the Layout Editor window, either manually using the  button to set the viewport or use the bookmark which has already been created.

3. Choose **View > Bookmarks**, then click *vco\_resistor* which changes the viewport of the *vco* cell.



4. You can also open the bookmark in a new window. Choose **View > Bookmarks > Manage** to invoke the Bookmark Manager.



5. In the Layout Editor window, fit the design using the bindkey **f**. Now double-click on the *vco\_resistor* line (outside of Name entry) in the Bookmark Manager. This opens the saved viewport in a new Layout Editor window.
6. Close the new Layout Editor window which was opened in the previous step.
7. Choose **File > Close** to close the Bookmark Manager.

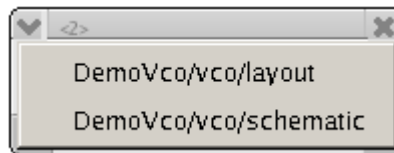
## Task 8. Window Navigation and Configuration

You may have noticed that there are multiple toolbars within the Custom Compiler design editors. You can configure the toolbars and the assistants anyway you like.

In this task, you will configure the toolbars for the Schematic Editor and assistants for the Layout Editor.

**Note:** Toolbars and assistants for other design editor windows can also be configured.

1. Choose **Design > Open Recent** from the Layout Editor window to open the *vco* schematic, which gives the following menu:

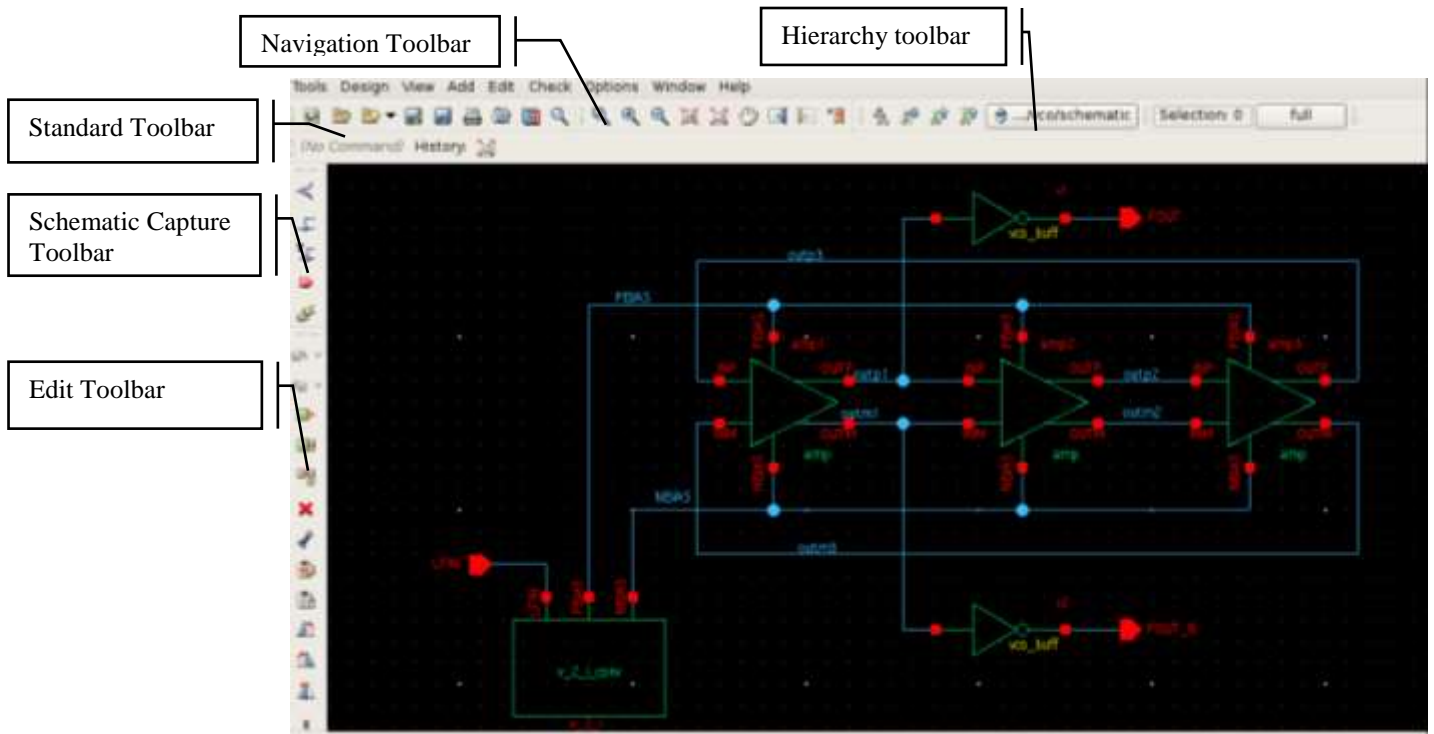


Select **DemoVco/vco/schematic** to open the cellview.

2. As seen in the following figure, there are different kinds of toolbars supported. Choose **Window > Toolbars > toolbar** to add or remove a specific toolbar.



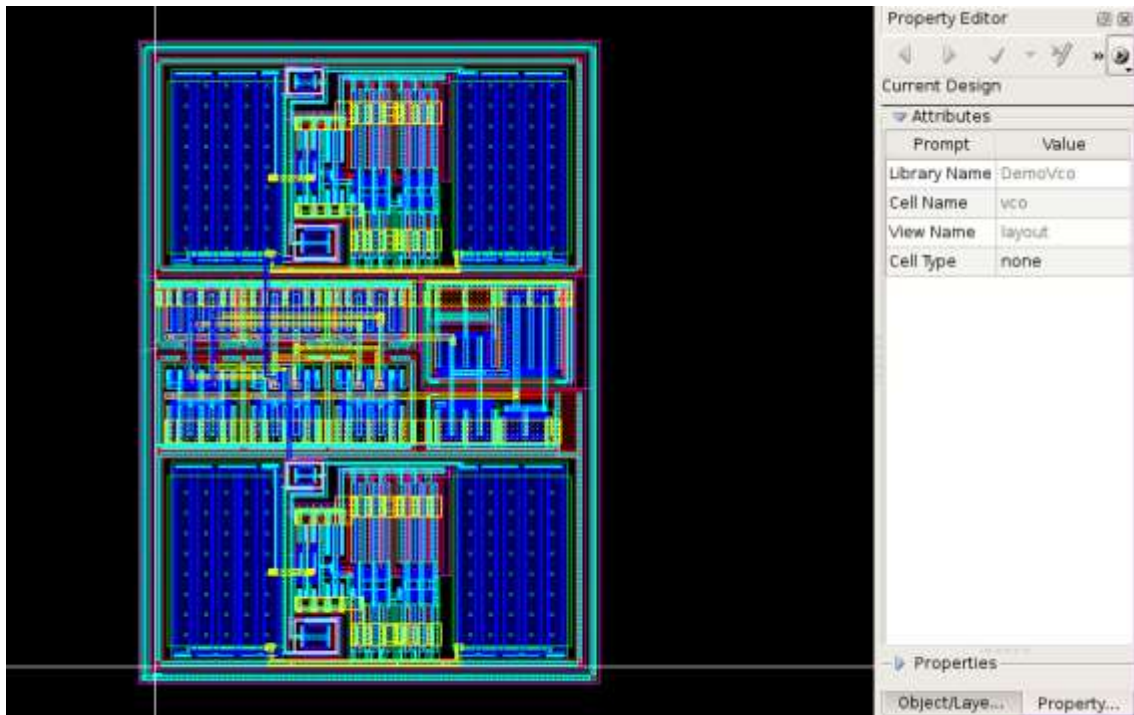
- a. Remove the **Check** toolbar shown on the left from the Schematic Editor.
- b. Change toolbar positions by dragging-dropping.




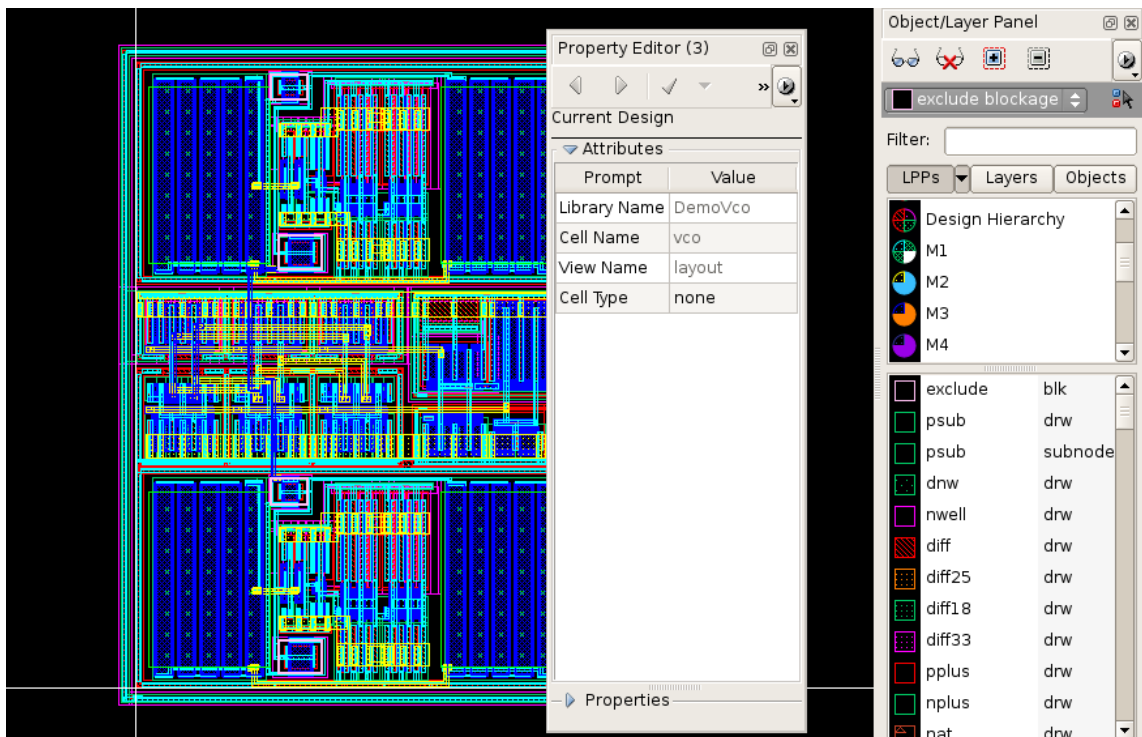
.....


**Note:** Docked assistants can also be moved around the window. They can be undocked and can be placed at a convenient location. Multiple assistants can be docked in the design editor window in different ways. For example, the **Property Editor** and **Object/Layer Panel** assistants can be arranged in the Layout Editor window.

3. Make the Layout Editor window active.
4. Choose **Window > Assistants** to access docked assistants. For example, add the **Property Editor** assistant to the Layout Editor window.



5. The **Property Editor** assistant can be undocked. To undock the assistant, click the undock  button in the **Property Editor** assistant. Now the assistant can be moved around.



6. Close the **Property Editor** assistant by clicking the close  button.



## Task 9. Selecting and Querying Objects

---

In this task, you will select, query, and deselect the objects in the canvas.

1. Make the Layout Editor window active.

**Note:** When cursor is on an object, the information about the object is displayed in **Object Info** toolbar. An example is given below.



2. Use **Ctrl+A** to select all objects in the window.

**Note:** You can select objects by clicking them.  
When no command is active, you can click and draw a rectangle to select all the objects within the rectangle.

3. Use **Ctrl+D** to deselect all the objects or click an empty area in the canvas.

**Note:** In a group of selected objects, objects which you do not want to select can be deselected by left-clicking them while pressing the **Ctrl** key.

4. Choose **Query > Query** and select any object in the layout. This returns the information about the selected object in the Custom Compiler Console window.

**Question 3.** What does the **Design > Summary** command return for the current *vco* design?

.....

## Task 10. Using the Transaction History Assistant

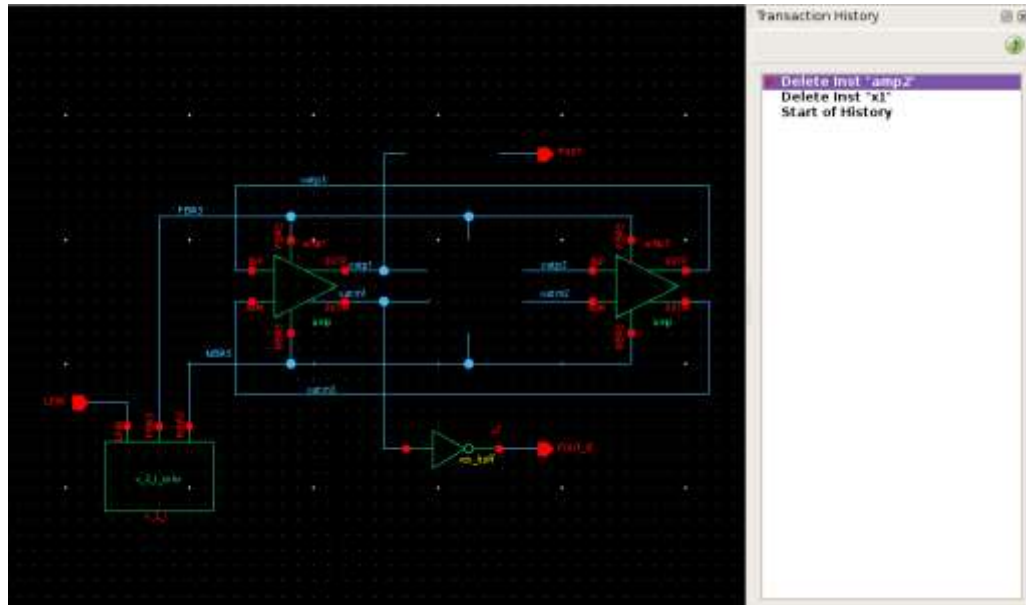
---

In this task, you will use the **Transaction History** assistant, a dockable assistant which displays a list of the most recent editing operations. You can also navigate the editing history.

1. Make the Schematic Editor window active.
2. Choose **Window > Assistants** to add the **Transaction History** assistant to the window. Fit the design.
3. Delete the instance of *vco\_buff* using the **Delete** key. It can be seen that the delete action is added to **Transaction History** as a transaction named *Delete Inst "x1"*.

**Note:** Observe that an asterisk is added to *DemoVco vco schematic\** (on the upper left) indicating that design was modified. This will be removed upon saving the design.

4. Delete the *amp2* instance. It can be seen that the delete action is added to Transaction History as *Delete Inst "amp2"*.



5. Click **Start History**. You can see that the *vco* schematic design has returned to its original state, that is, the *amp2* and *x1* delete actions are undone.

**Note:** Selecting a displayed transaction name will undo the action in the edit session

6. To redo the deletion of *x1*, click *Delete Inst "x1"* above **Start History**. To redo the deletion of *amp2*, click the last *Delete Inst "amp2"* in the history.

**Note:** Selecting a greyed out transaction name will redo the action in the edit session.

7. Click **Start History** to restore the design to its original state.
8. Close the **Transaction History** assistant.

## Task 11. Using the Display Resources Editor GUI

The Display Resources Editor [DRE] is used to control the visual attributes of the objects displayed in the design editor window.

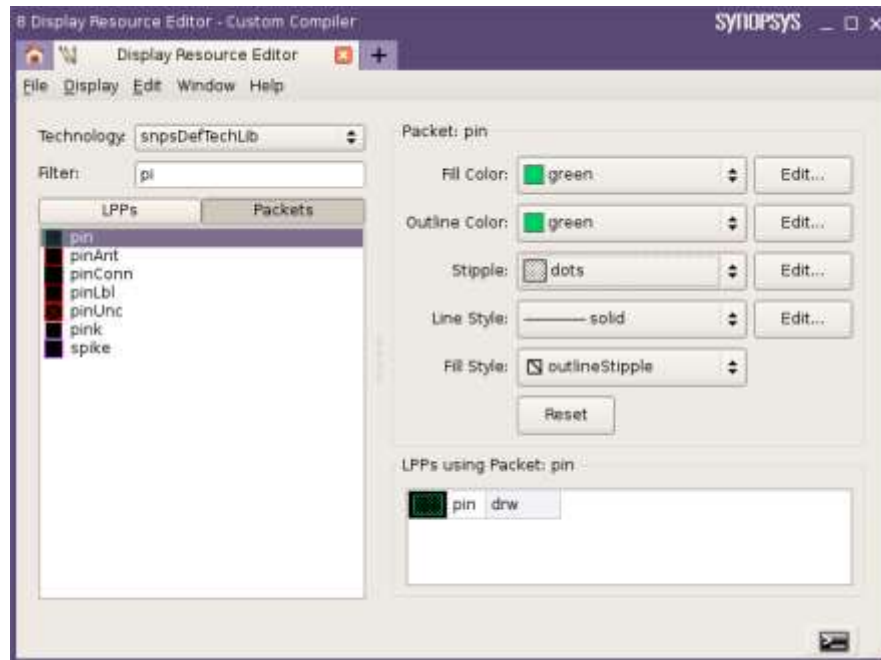
In this task, you will change the display attributes of the schematic pin objects.

1. Choose **Tools > Display Resource Editor** from the Console window to invoke the Display Resources Editor GUI.

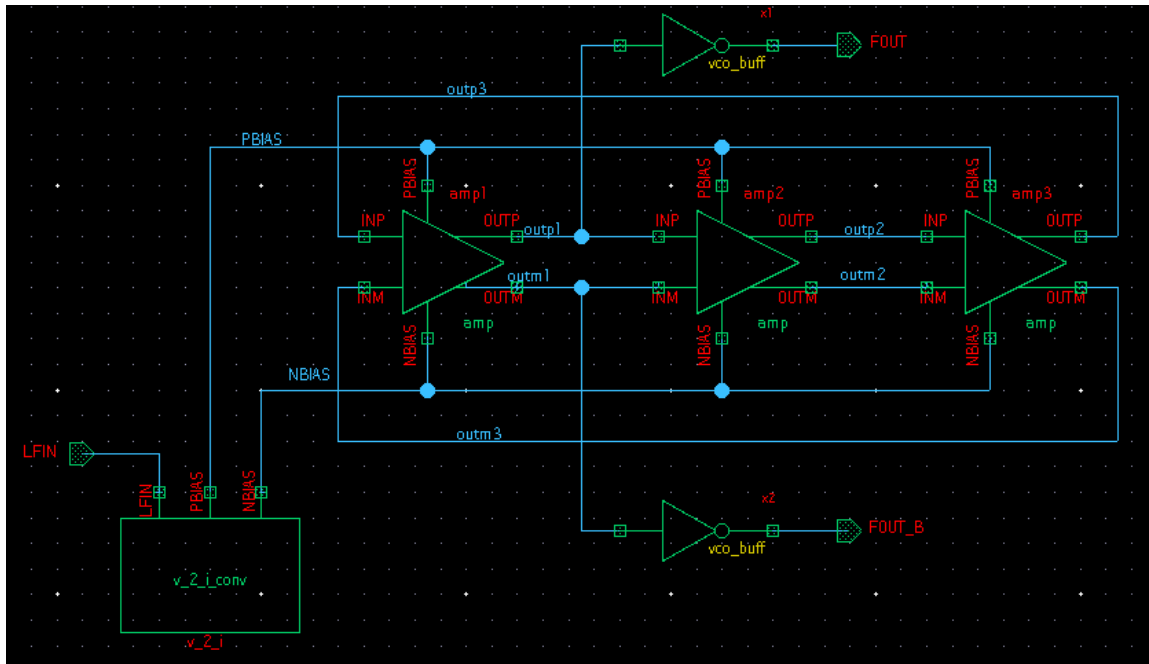
**Note:** The default settings are loaded from the *display.tcl* file shipped with the Custom Compiler platform. The settings are overridden, if there is *display.tcl* at any other standard startup location.

2. Make the Schematic Editor window active. The DRE now lists the packets and packet aliases defined, under the **Packet** browser.

3. In the **Filter** box, enter '*pi*'. Only the packet names that contain “pi” are listed now.
4. Select **pin** from the list of packets. This packet is used for the pins in the Schematic Editor window.
5. Observe that **Display Attributes** in the DRE GUI is changed to show the packet definition for the **pin**.
6. Under **Display Attributes**, select **green** as **Fill Color** and **Outline Color**. Set the **Stipple** pattern to **dots**.



7. Choose **File > Save As** to save the current settings to a new file *display\_new.tcl*.
8. Make the Schematic Editor window active. Use the bind key **f** to fit the design. Observe that in the Schematic Editor window, the display attributes of the pin have changed as shown in the following figure.



**Note:** This change is valid for the current session only. For future sessions, the saved *display\_new.tcl* file must be loaded to inherit the saved settings.

9. Choose **File > Close** to close the DRE.
10. Close the Schematic Editor and Layout Editor windows.

## Task 12. Exiting the Custom Compiler Platform

1. From the Home page choose **File > Quit** to quit the Custom Compiler platform. At the **Exit** confirmation window, specify whether you want the state to be saved. Alternatively, you can type *exit* in Console Window.
2. Check that in the *synopsys\_custom* directory, *history.xml* file has been created, which has the list of cells that was opened in the session.

**Note:** Created bookmarks would be written in *bookmarks.xml* file at session termination.


If the **Save Preferences** option is selected in the **Exit** dialog, then preferences modified in the session would be written to *prefs.xml* file, in *synopsys\_custom* directory.

If **Display Resources** is checked, the currently configured display resources are saved into *display.tcl* in the *synopsys\_custom* directory. This operation is incremental in that only modified display resource settings are saved.

# Answers / Solutions

## Task 1. Using the Preference Manager

**Question 1.** Save the preferences from **General Options** to a file *test.xml*.

In the **General Options** window, click the  button, next to the **Defaults** button. Select **Save** from the menu. This brings up the **Save Preferences** dialog. Enter the file name as *test.xml* in the **File Name** field. Click **Save** to save the preferences to test.xml.

## Task 4. Finding Commands

**Question 2.** Get the syntax of the command **dm::createLib**.

Enter *dm::createLib -help* in the Console window. It should return the following.

```
> dm::createLib -help
Syntax:
dm::createLib <string> -path <string> [-dmSystem
<string>]
Arguments:
name :      Library name
-path :     Location of the library
-dmSystem : DM system for the library (Default is:
oaDMFileSys)
Returns:
<oaLib> : New library
```

## Task 8. Selecting and Querying Objects

**Question 3.** What does the **Design > Summary** command return for the current *vco* design?

It returns the object types and counts of each object types in the design. For instances, it returns the master library and instance names.

# 2

## 2. Library Manager

### Learning Objectives

During this lab, you will use the Library Manager to create Library, Cell, and CellViews. In addition, you will use the editing functions to manage the Library, Cell, and CellViews.

After completing this lab, you should be able to use Library Manager to

- Create Libraries, Cells and CellViews
- Create and manage cell categories
- Editing functions to manage library, cell, and cellviews
- Handle technology data



**Lab Duration:**  
30 minutes

# Introduction

In this lab, you are provided with the technology files and sample OpenAccess libraries. You will create OpenAccess library, cell, and cellviews. You will also handle technology data, and manage cell categories.

# File Locations

All files for this lab are located in the directory **Library\_Manager\_Lab1**

## Directory Structure

|                             |                           |
|-----------------------------|---------------------------|
| <b>PDK</b>                  | PDK directory             |
| <b>Library_Manager_Lab1</b> | Current working directory |
| <b>ref_data</b>             | Reference data directory  |
| <b>lib.defs</b>             | library definition file   |
| <b>PLL</b>                  | PLL library               |
| <b>PLL_TB</b>               | PLL testbenches library   |
| <b>techfiles</b>            |                           |
| reference40nm.tf            | Sample Technology file    |
| display.tcl                 |                           |
| display.drf                 |                           |
| reference40nm_layer.map     |                           |

## Answers & Solutions

Each lab contains answers to all questions and results or solutions.

*You are encouraged to verify your results by checking the **Answers/Solutions** section at the end of each lab.*



# Instructions

## Task 13. Start Custom Compiler

---


3. In the Unix window, change your working directory to **Foundation\_Lab2** and invoke Custom Compiler.

```
Unix% cd Library_Manager_Lab1
Unix% custom_compiler &
```

## Task 14. Create the Test Library

---

In this task, you will create library *TestLibrary*.

4. Invoke **Library Manager** from the Home Page.
5. From the **File** menu, select **New > Library** or click the New Library button .
6. At the **Name** field enter *TestLibrary*.
7. Select the option **Import File** for **Technology** and use the browse button to select the technology file.
8. From the directory *ref\_data/techfiles*, select the tech file *reference40nm.tf* and click OK button in **New Library** dialog. This creates the library named *TestLibrary* and added to **Libraries** and a directory in the name of TestLibrary has been created in the current working directory.

**Note:** Library also can be created by activating CSM. Right-click in the **Libraries** to activate the CSM and select **New** from the menu.

```
Unix% ls
lib.defs  ref_data  synopsys_custom  TestLibrary
```

9. Open the **lib.defs** file by vi editor; it should have the definitions like this.

```
Unix% cat lib.defs
DEFINE TestLibrary ./TestLibrary
```

## Task 15. Using INCLUDE file

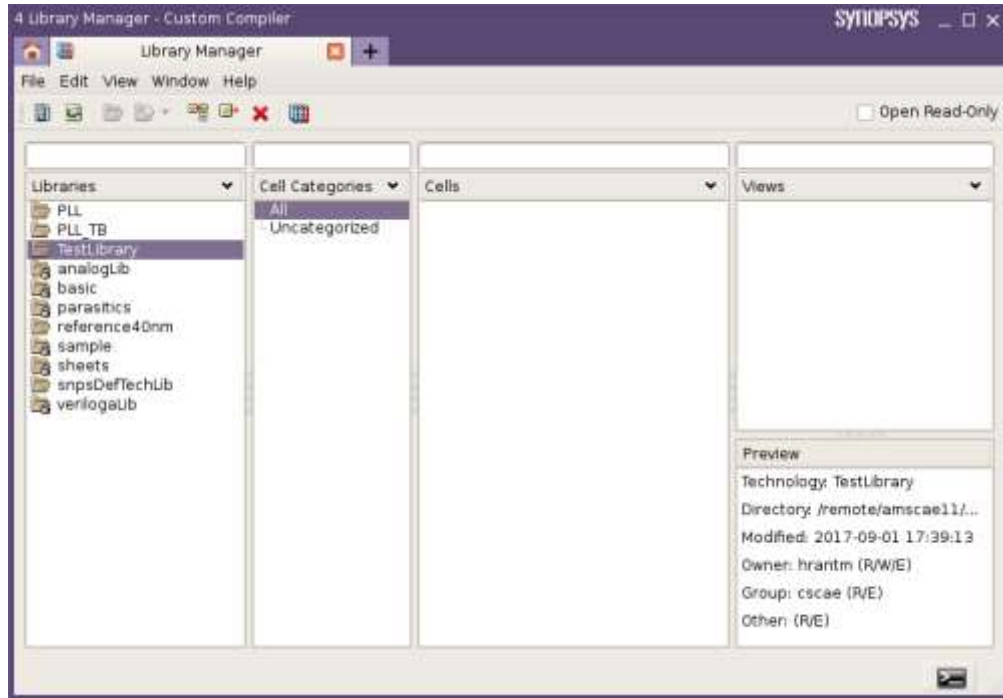
---

In this task, you will add the libraries by editing the lib.defs file.

10. Add the following lines in the *lib.defs* and save it.


```
INCLUDE ../ref_data/lib.defs
```

11. In the **Library Manager**, select **File → Refresh** to see the included libraries, which should resemble the below given figure.



## Task 16. Create a layout cell view

In this task you will create layout.

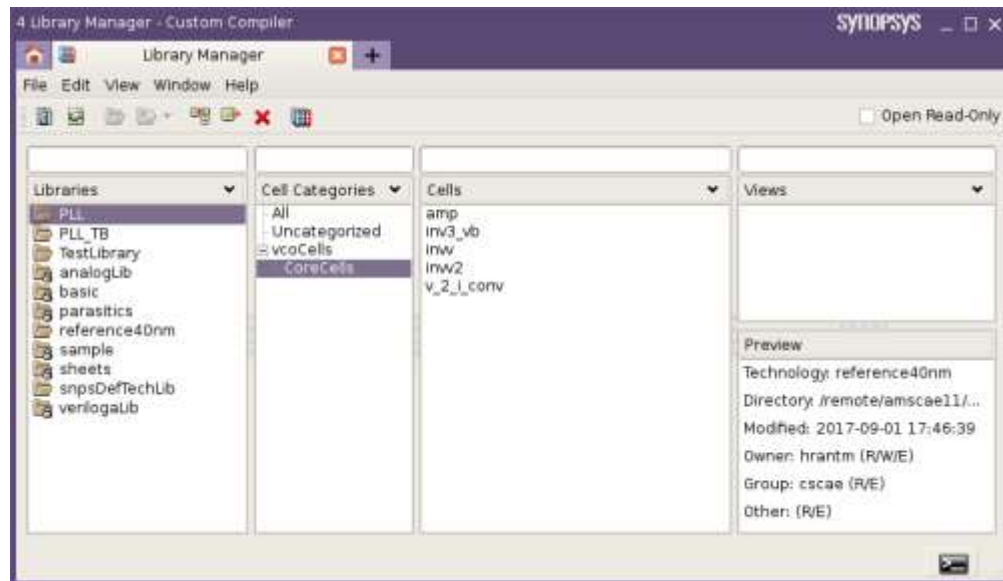
12. Make the **Library Manager** window active. Select the library *TestLibrary* from **Libraries**.
13. Invoke Create **CellView** window from the **File > New > CellView** or clicking the  button.
14. In the **Cell Name** field enter **test**.
15. In the **View Name** field, select **layout**.
- Note:** Observe that selecting layout automatically changes the **Editor** to **Layout Editor**.
16. Make sure that **Open on Create** option is selected and click OK. This creates the layout view for the cell *test* and the design is opened in a new window.
17. Close the *test* cellview.

## Task 17. Managing Cell Categories

---

In this task, you will create and modify cell categories using the **Library Manager**.


18. Select the library PLL.
  19. Create a new cell category called *CoreCells* as follows:
    - a. From Cell Categories view list CSM select New... menu item. Type *CoreCells* and press **OK**. This creates *CoreCells* category and added to the **Cell Categories** list.
  20. Create another Cell Category called *vcoCells*.
  21. Create *CoreCells* category members
    - a. Select Uncategorized category from Cell Categories view list.
    - b. Select following list of cells *amp*, *invv*, *invv2*, *inv3\_vb*, *v\_2\_i\_conv*, drag and drop into the *CoreCells* category. They will move from Uncategorized category to the *CoreCells* category.
- Note:** Multiple cells can be selected using Ctrl+click. The list of selected and moved cells are now the members of *CoreCells* cell category.
22. Select *CoreCells* category and drag-drop into the *vcoCells* category. *CoreCells* is now a sub-category of the *vcoCells* category.

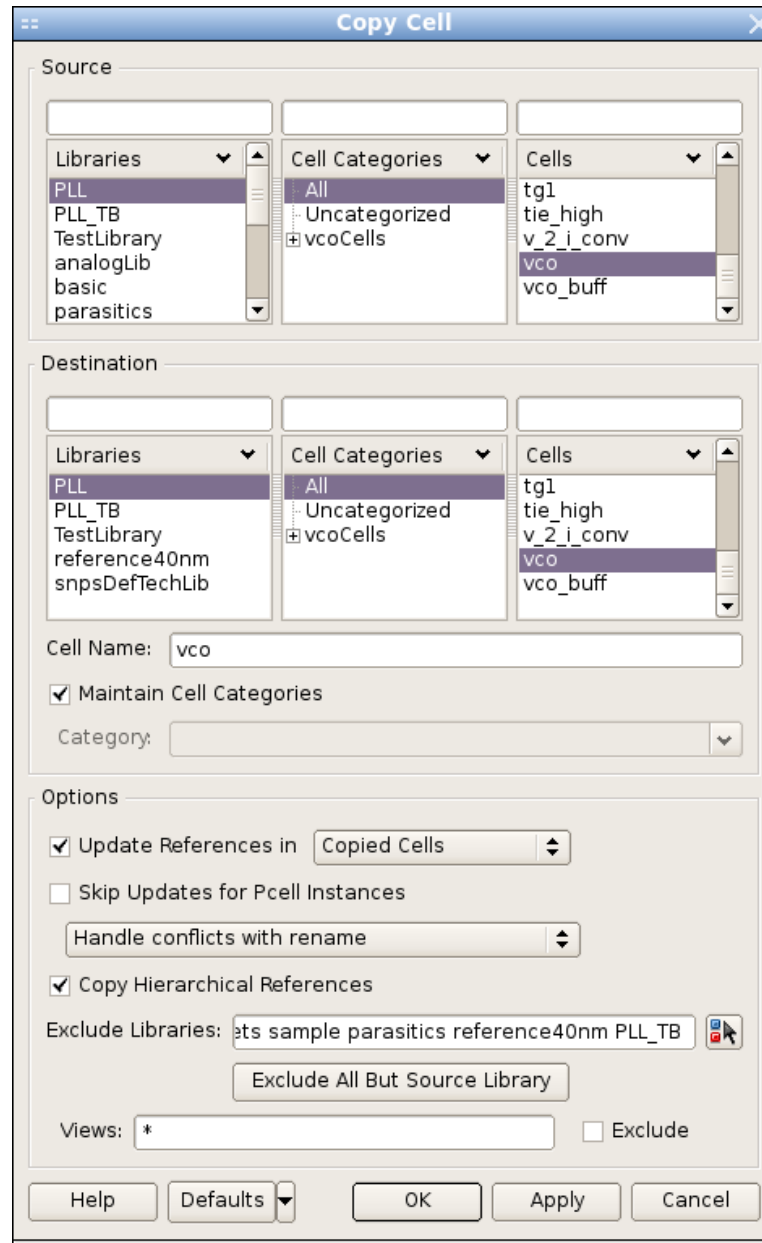


---

## Task 18. Copy Cells

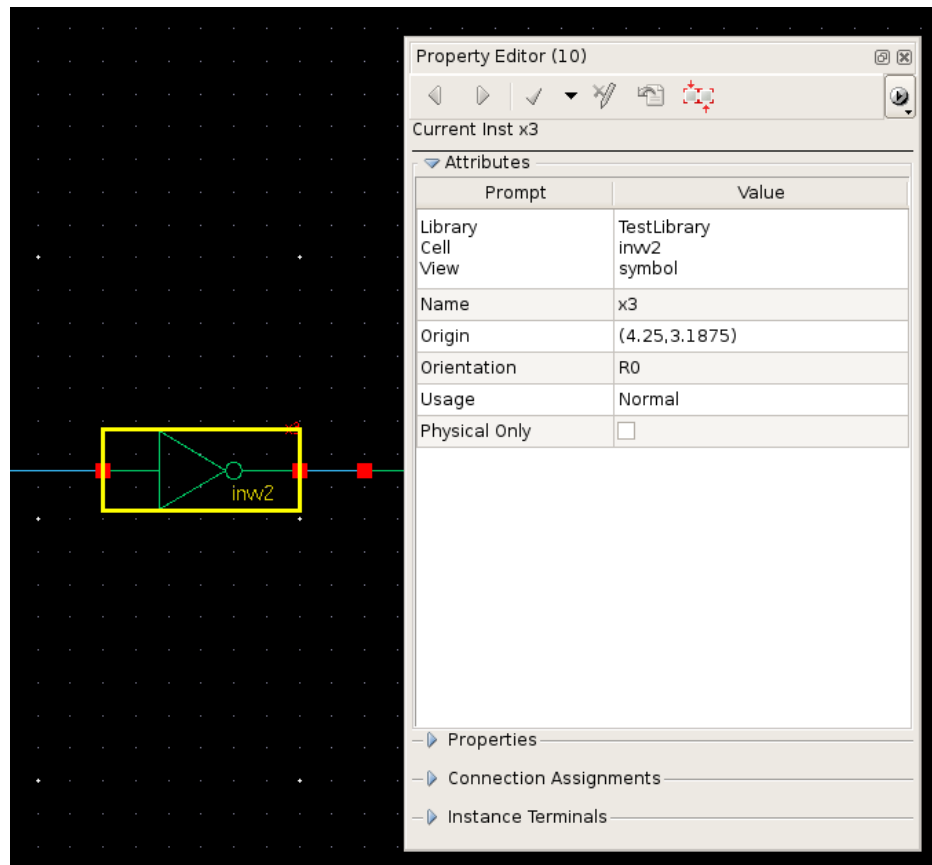
In this task you will use one of the editing functions of the library - to copy a list of cells from a library.

23. Open the **Copy Cell** window, from PLL/vco cell CSM or click the  button.
24. In the **Destination** area, select **TestLibrary**.
25. Select **Copy Hierarchical References** check box.
26. Exclude all reference libraries except PLL.
  - a. Open Exclude Libraries dialog, by clicking push button next to the Exclude Libraries entry and select libraries to exclude.
27. Click OK



28. Open TestLibrary/vco/schematic

29. Bring up Property Editor (bindkey “q”)
30. Select any inverter instance
31. Observe reference



- Question 4.** Repeat copy operation of the vco cell, sequentially using the following options. See what happens with instance references.
- i. Do not update references. Do not rename conflicting cells in destination library.
  - ii. Update references in Destination Library and rename conflicting cells.



# Answers/Solutions

## Task 9. Copy Cells

1. What happens with instance references?

| case   | Rename<br>Conflicting<br>Cells | Update<br>References   | Result   |
|--------|--------------------------------|------------------------|--|
| P 7.   | No Conflict                    | Copied Cells           | vco/schematic instances point to copied cells in the TestLibrary   |
| Q.1.i  | Don't<br>Rename                | None                   | All cells are rewritten, vco instances point to cells the PLL library  |
| Q.2.ii | Rename                         | Destination<br>library | Both, vco/schematic and vco_PLL/schematic instances point to the new copied(renamed) cells in destination library. |

# 3

## 3. Schematic Entry

### Learning Objectives

The main objective of this lab is to familiarize you with the Custom Compiler Schematic Environment and allow you to learn the capabilities of the Custom Compiler Schematic Editor by capturing the schematic of an analog circuit.

After completing this lab, you should be able to:

- Create and place Instances
- Edit using editing functions
- Edit the instance parameters
- Create Pins
- Create Wire Objects
- Create Wire Names
- Check and Save the Design



**Lab Duration:**  
45 minutes



# Introduction

During this lab, you will create a schematic design of an analog circuit, a CMOS Differential Amplifier, following a systematic design methodology using various features of Custom Compiler Schematic connectivity objects, wiring, and editing functions.

In this lab, you are provided with *lib.defs* file which contains the mapping of *reference40nm* (Reference design library), *analogLib*, *verilogaLib*, *basic*, *sample* and *sheets* libraries (libraries shipped with the installation). These libraries contain the primitive components (passive and active elements) and other design components to build the design. You will use these libraries as reference libraries to build your schematics.

Design creation is generally done in two steps: first, the setting of design specifications and flow, second, the implementation and verification with EDA tools. Before creating the schematic you need a set of specifications as a target to create the CMOS Differential Amplifier design. At the same time, it is important to understand the use model of the commands that are required to build it.

Please see the section “Data Creation and Editing” of the *SE Command Reference Manual* for the commands that are used in this lab.

## A. Design Specifications

---

### Definition of Important terms:

**Slew rate:** The maximum rate at which the output voltage changes when the input signal is large.

**Input common-mode range:** The range of common-mode voltage values over which the differential amplifier continues to sense and amplify the differential-mode signal.

**3db-Bandwidth:** The frequency at which the gain of the amplifier goes below 3dB from the unity frequency.

**Phase margin:** The difference between the value of the phase at the frequency in which the magnitude of loop gain is equal to unity and -180 degrees.

**Power dissipation:** The product of total current drawn from the power supply and the applied supply voltage

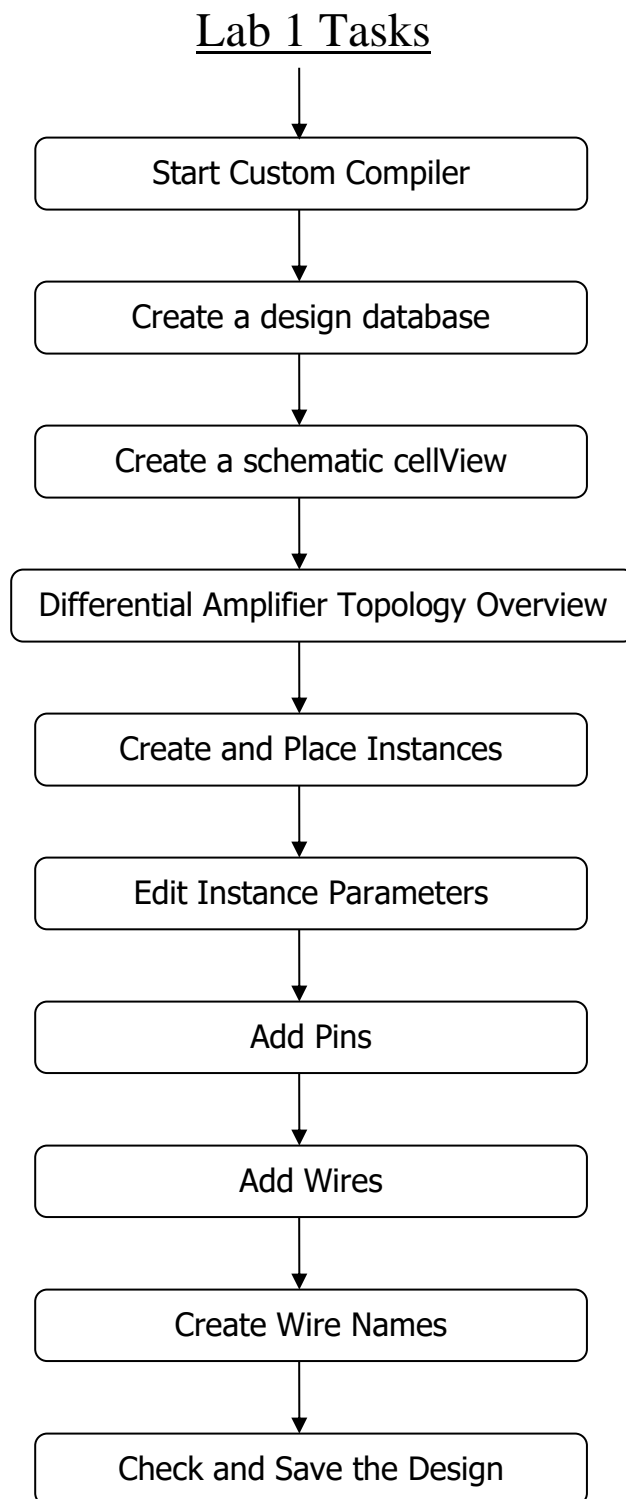
The specifications for the differential amplifier consist of:

- Small-signal gain  $A_v$
- Frequency response for a given load capacitance,  $\omega_{-3dB}$
- Input common-mode range (ICMR) or maximum and minimum input common-mode voltage [ $V_{IC}(\text{max})$  and  $V_{IC}(\text{min})$ ]
- Slew rate for a given load capacitance, SR
- Phase Margin, PM
- Power dissipation,  $P_{\text{diss}}$

In this lab, you will design a differential amplifier with the following specification:

|                   |   |
|-------------------|---|
| $A_v$             | $\geq 40$                                       |
| $\omega_{-3dB}$   | $\geq 100 \text{ kHz}$ ( $C_L = 5 \text{ pF}$ ) |
| ICMR              | $-0.8V \leq \text{ICMR} \leq 0.8V$              |
| SR                | $\geq 5V/\mu s$ ( $C_L = 5 \text{ pF}$ )        |
| $P_{\text{diss}}$ | $\leq 0.3mW$                                    |
| PM                | $\geq 85^\circ$                                 |

# Flow Overview



# File Locations

All files for this lab are located in the directory *SE\_Schematic\_Entry\_Lab1*.

## Directory Structure

*SE\_Schematic\_Entry\_Lab1*

Current working directory

*ADC*

OpenAccess Design library

## Relevant Files

*lib.defs*

Library definitions file

## Answers & Solutions

There is an *ANSWERS / SOLUTIONS* section at the end of this lab. You are **encouraged** to refer to this section often to verify your answers or to obtain help with the execution of some steps.

# Instructions

The goal is to successfully design a Differential Amplifier circuit using the capabilities of the Custom Compiler Schematic Editor. Follow the instructions in the tasks below to accomplish this goal, but make sure you try or play a little with the commands in order to increase the probability of generating questions while you are in the class and deepen your understanding of the tool.

## Task 19. Start Custom Compiler

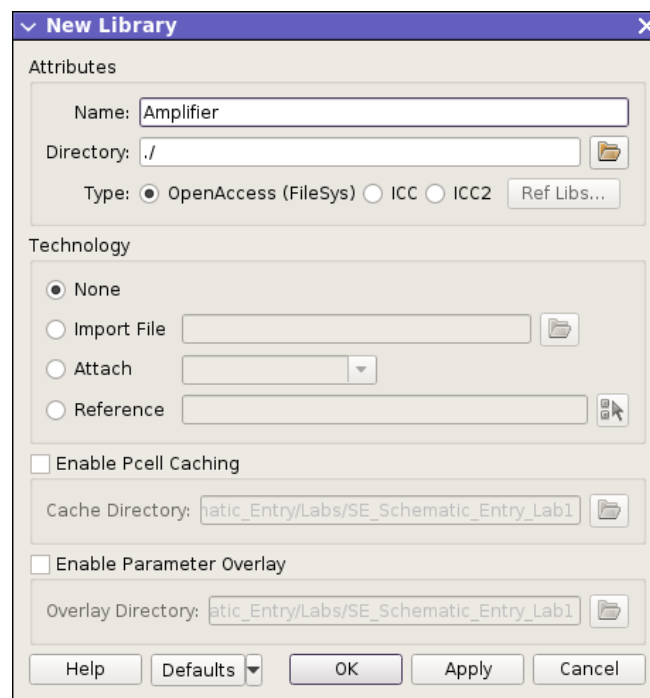
32. In the Unix terminal window change your current working directory to *SE\_Schematic\_Entry\_Lab1*. This will be your working directory for this lab.
33. Start Custom Compiler from the Unix prompt.

```
custom_compiler &
```

## Task 20. Create a design database

A library (OA design database) is a logical collection of data representing a design. It is the top level entity of the design data (schematic, symbol, netlist and so on).

34. Create a new design library *Amplifier* in **FileSys** mode and attach the technology library *reference40nm* using **File→New→Library...** in Library Manager window



## Task 21. Create a schematic cellView

---

The schematic cellView describes an individual building block of a chip or a system which contains the actual design. (In this lab it is a Differential Amplifier design).

35. In the design library *Amplifier*, create a schematic cell *Diff* with the view name *schematic*.

**Note:** The *Diff* cellView will be opened in a Schematic Editor window. In the Library Manager, a new cell name *Diff* is added in the **Cells** column and a new view name *schematic* is added in the **Views** column.

## Task 22. CMOS Differential Amplifier Topology Overview

---

The differential amplifier is one of the most versatile circuits in analog circuit design. Fig.1 shows the CMOS Differential Amplifier schematic for the target specification that uses:

- n-channel MOSFETs *M0* and *M1* to form a differential pair
- Two p-channel MOSFETs *M2* and *M3* at the top forming the current mirror load and
- The current sink implemented by n-channel MOSFETs *M4*, *M5* and a RESISTOR *R1* to bias *M0* and *M1*.

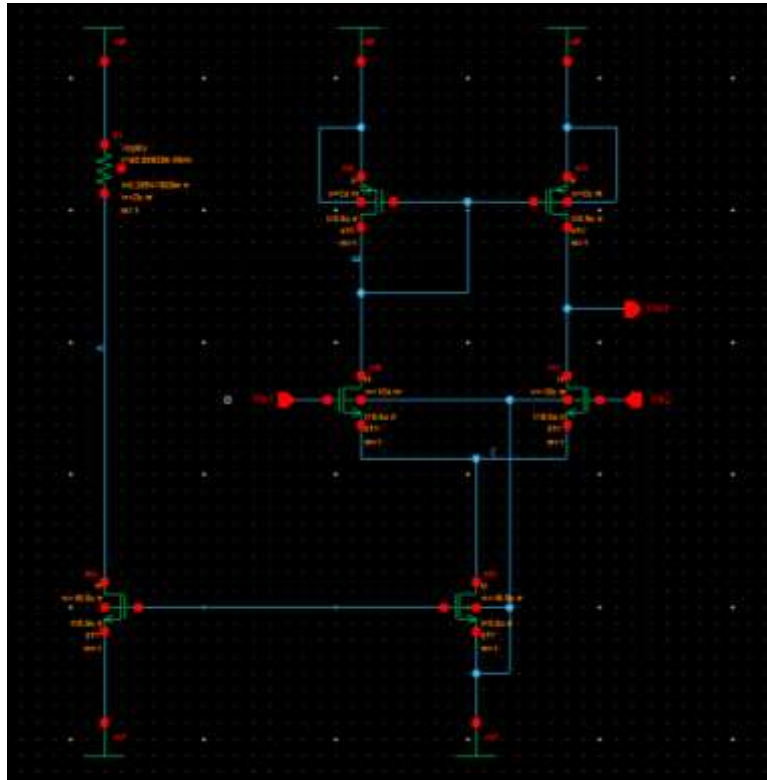


Fig.1

The dimensions of the elements as per the target specifications are listed below:

| Primitive            | Instance Name | Parameters |        |
|----------------------|---------------|------------|--------|
| n_4t (nmos)          | M0            | w=12u      | l=0.5u |
| n_4t                 | M1            | w=12u      | l=0.5u |
| n_4t                 | M4            | w=18.5u    | l=0.5u |
| n_4t                 | M5            | w=18.5u    | l=0.5u |
| p_4t (pmos)          | M2            | w=2u       | l=0.5u |
| p_4t                 | M3            | w=2u       | l=0.5u |
| rnpoly<br>(resistor) | R1            | r=60k Ohms |        |


You will create a Differential Amplifier schematic with the above dimensions. The guidelines to achieve this task will be provided in the subsequent tasks.

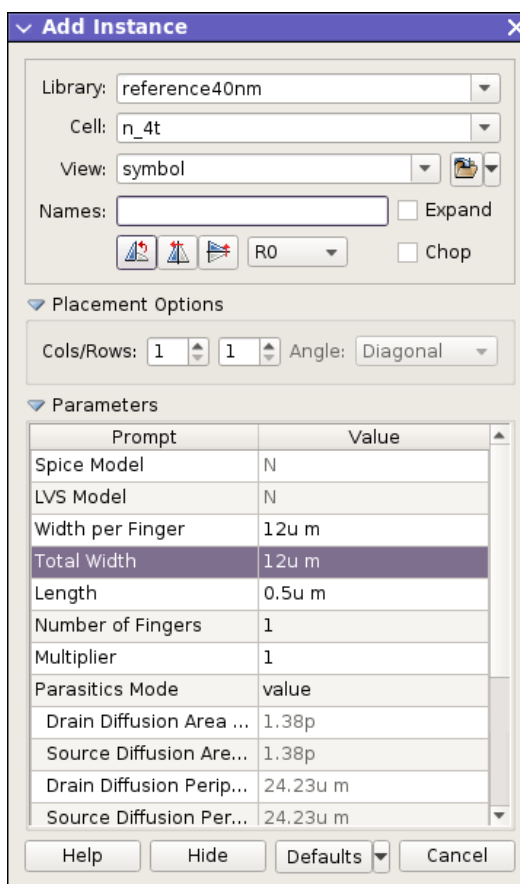
## Task 23. Create and Place Instances

---

The first step in creating the schematic after Library/Cell/View creation is adding the instances. Instances are the cellView objects referencing other symbols or schematics.

In this task, you will learn to add the instances with the default and overridden parameters using the Task 4 schematic as a reference to build the CMOS Differential Amplifier schematic.

36. Add the *n\_4t* (nmos) instance *M0* of parameter  $w/l = 12u/0.5u$  from the library *reference40nm* using **Add → Instance (i)...** menu command or by clicking on the Add Instance  button.



**Note:** Here **w** refers to width and **l** refers to length.

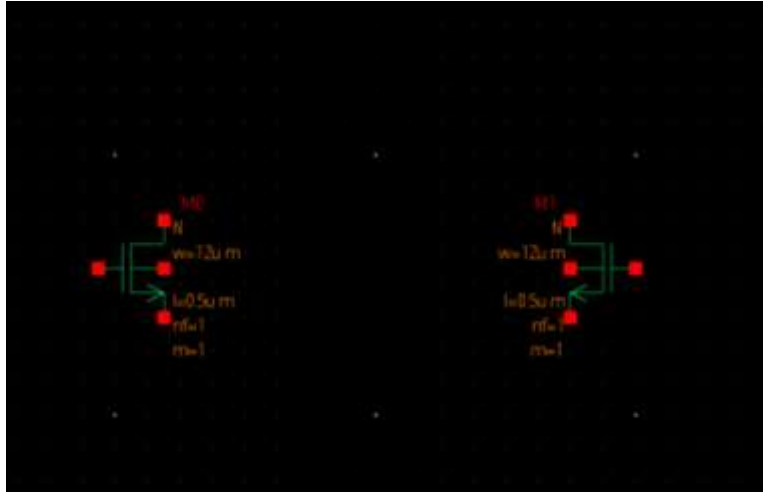
When an instance (complete Library/Cell/View) is selected in the **Add Instance** dialog and the mouse cursor is in the canvas, the image of the instance of the selected cellView (symbol or schematic only) will appear.

When the instance is placed, a set of parameters and attributes are also displayed on the canvas.

37. Copy and mirror the instance *M0* and place it on the schematic canvas to form the Differential pair as shown in the image below:

**Note:** When you copy and place the instance, the tool automatically generates the name for the copied instance. In this case it will be *M1*.





38. Similarly, place the other instances  $M2$  ( $p\_4t$ ),  $M3$  ( $p\_4t$ ),  $M4$  ( $n\_4t$ ),  $M5$  ( $n\_4t$ ) and  $R1$  ( $rnpoly$ ) with their default parameters.

**Hint:** Use “**Device Palette**” assistant for quickly insertion of instances. To open the assistant, choose **Window** → **Assistants** → **Device Palette**.

**Note:** Always make sure that you are not confusing a PMOS transistor with an NMOS transistor. The differences between the PMOS symbol and the NMOS symbol are:

- There is normally a tiny circle at the gate terminal of PMOS transistors.
- The direction of the arrow which marks the *source* terminal of the transistor is different. The arrow points from the *source* terminal towards the transistor in the PMOS symbol, while it points out towards the *source* terminal in the NMOS symbol.

**Question 4.** How can multiple instance names be entered in the **Add→Instance** form?

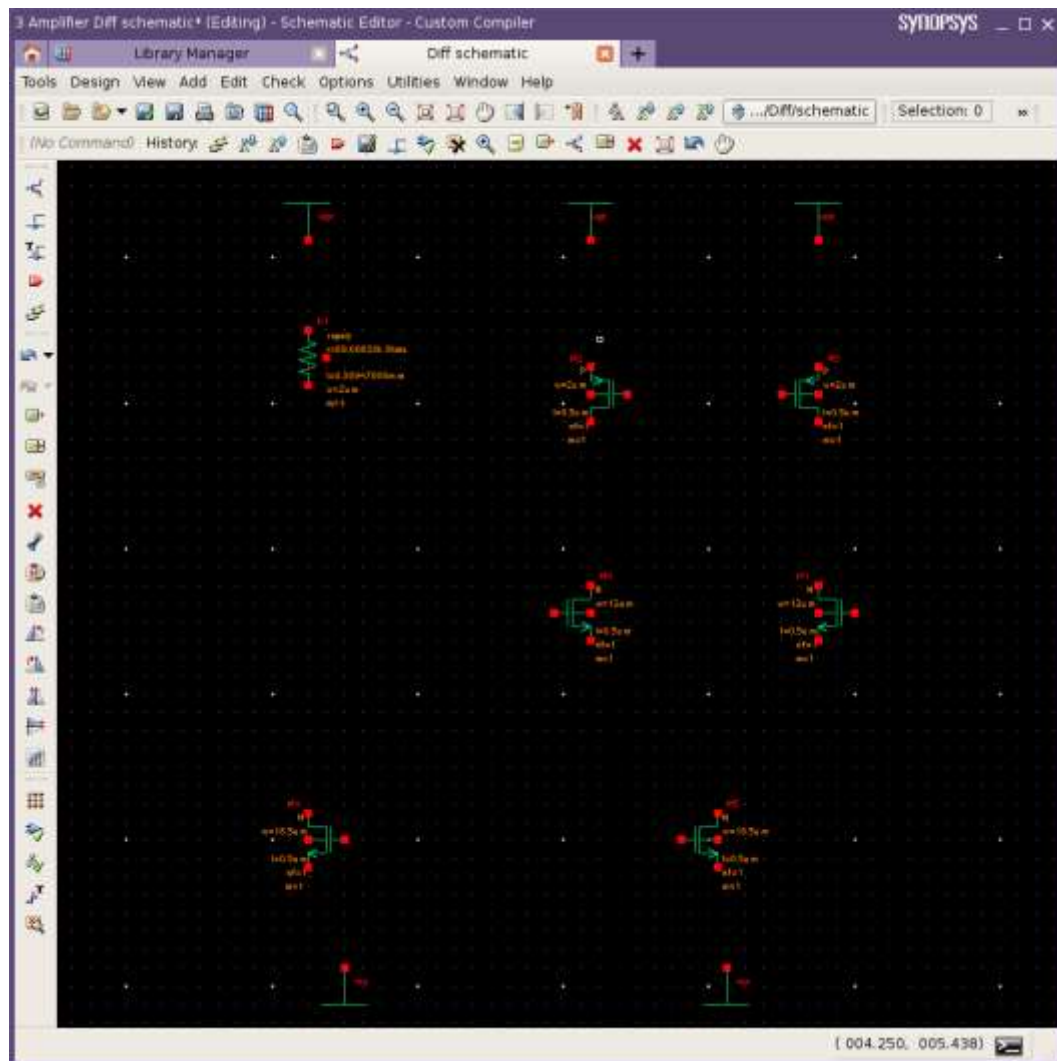
39. After placing primitive elements, you need to add the supply voltages to the schematic. Supply voltages are placed in the same manner as instances are placed in the earlier task.

- a. Place the global supply nets  $vdd$  and  $vss$  from the library *analogLib*.

**Note:** Global nets/symbols can be used to electrically connect to other global nets with the same name anywhere in the hierarchy without going through instance terminals.

$vdd$  and  $vss$  define the global power.

40. After the initial placement, move the instances to adjust their positions as per your choice.
41. At this point, the schematic should look like this:



## Task 24. Edit Instance Parameters

Instance parameters are an important aspect of the design and they should be defined properly to come as close to the design specifications as possible.

In this task, you will learn to edit the parameters values of the instances to follow the values as defined in task 4.

42. Change the parameter value of the resistor  $R1$  using **Edit → Properties → Property Editor** menu command or by pressing the bind key **q**.

- a. Resistance value to 60k Ohms

**Question 5.** How are the default values of the parameter displayed in the Property Editor assistant?

43. Similarly, change the parameter values of the *nmos* instances *M4* and *M5* simultaneously (**Hint:** Use Shift-click to select the multiple instances).
- Width value to 18.5u
  - Length value to 0.5u
44. Parameter values can also be changed using the on-canvas editing feature of Custom Compiler. Change the parameter values of the *pmos* instances *M2* and *M3* by using the Middle-click on the parameter values displayed on the canvas. Follow these steps:
- Select **Edit...** from the Middle-click menu.
  - Edit the parameter value.
    - Width value to 2u
    - Length value to 0.5u
  - Press **Enter** or click on the canvas to apply the changes.
- Note:** The other ways of activating on-canvas editing feature are selecting the parameter and pressing enter or just double clicking on that parameter.

## Task 25. Add Pins

---

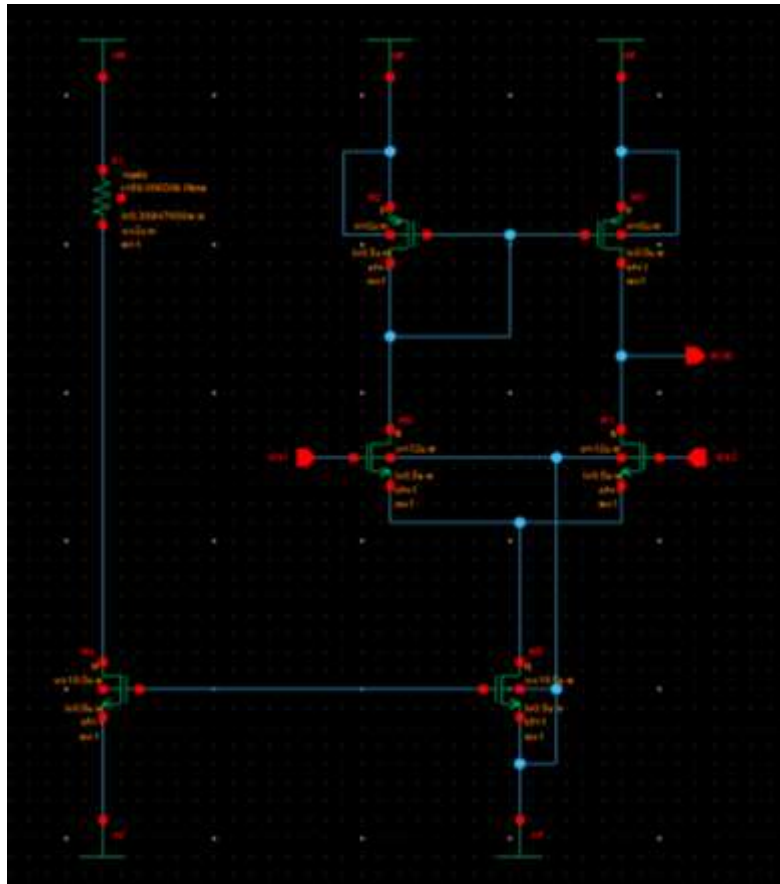
After all the components are placed with their parameter values set, the next step is to create pins to define the input and output signals of the circuit.


In this task, you will learn to add pins using **Add→ Pin**.

45. Create an input pin with name *Vin1* and *Vin2* to connect to the gate of *M0* and *M1* respectively.
46. Create an output pin with name *Vout* to connect to the drain of *M1* and *M3*.



47. Your schematic should look like the schematic shown in the image below:



48. By now you have successfully learnt to place the instances, pins and edit the instance parameters. Save the *Diff* schematic design using **Design → Save** or by clicking on the Save  button.
49. Close the *Diff* schematic window using **Design → Close**.

## Task 26. Useful Concepts – I

---

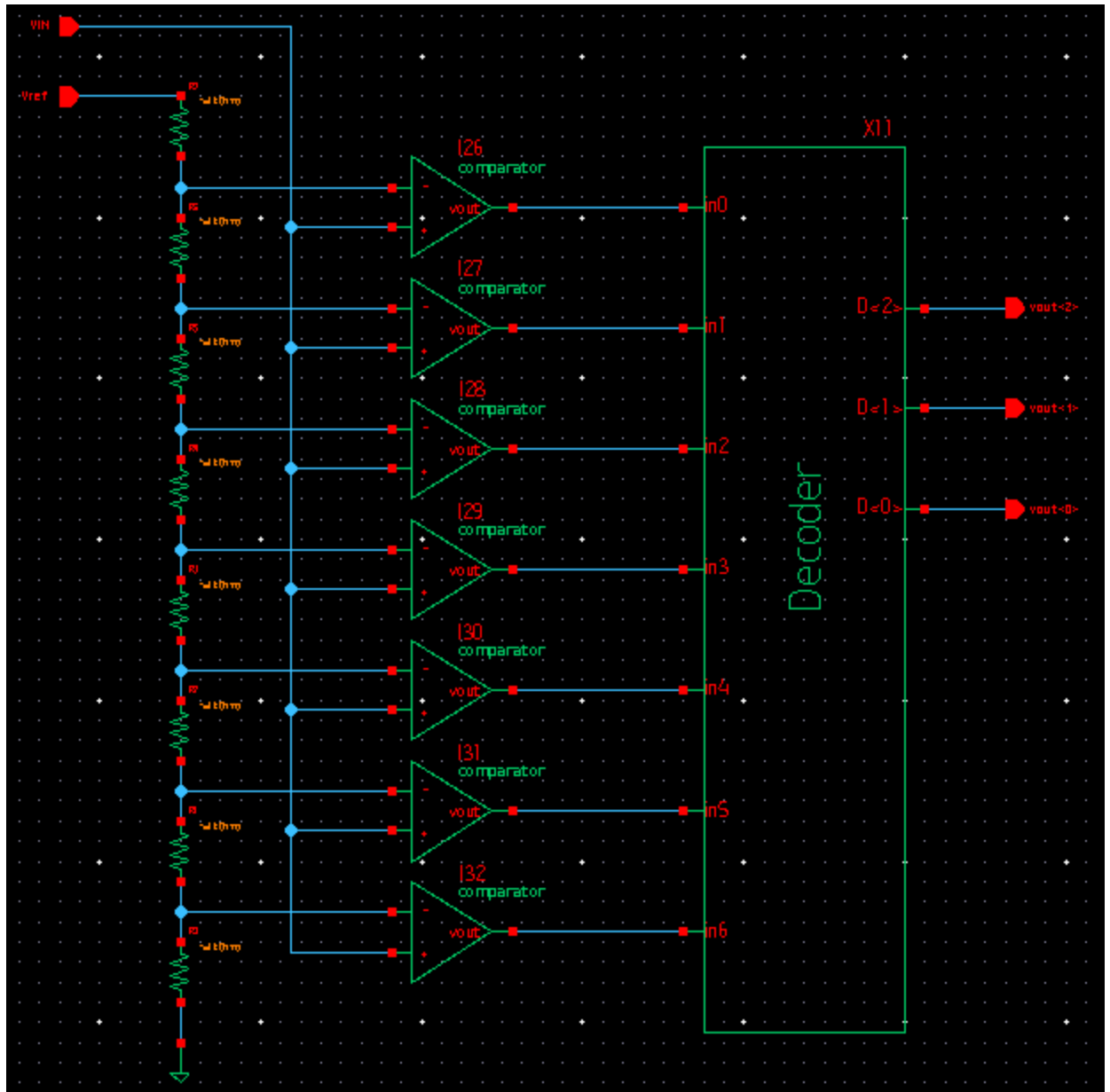
You have now learnt how to place and edit the instances and pins. In this task you will learn some useful techniques which will help you to create more complex designs with ease.

You will learn these concepts by completing 3-bit Flash ADC following the below instructions.

50. From the design library *ADC*, open the schematic cell *3bit-Flash* of the view name *schematic*.

**Note:** The *3bit-Flash* cellView will be opened in a Schematic Editor window showing the *3bit-Flash* schematic.

51. The figure shows the complete schematic of *3bit-Flash* ADC. You will place the design components of this schematic, using different techniques in subsequent steps.

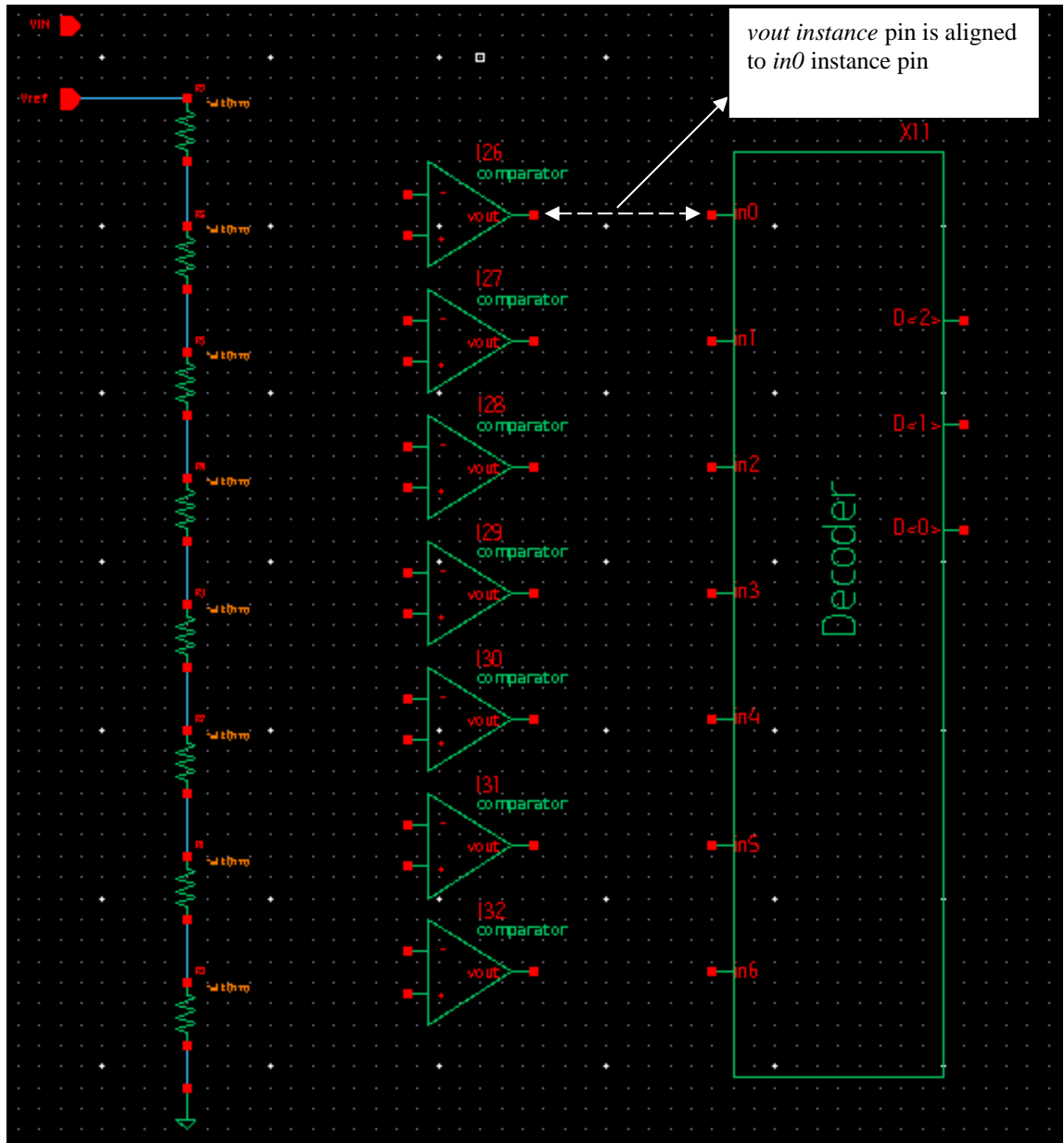


52. Place all the *comparator* instances as shown below from the library *ADC* using **Add → Instance...** menu command in two mouse clicks. Follow these steps:

- Set the **Cols** field to **1** and **Rows** field to **7**
- Place the first instance on the Canvas such that *in0* terminal of Decoder *X11* is aligned to *vout* terminal of the comparator as shown in the fig.

**Note:** As you place the first instance, you will observe that the remaining 6 instances appear on your mouse cursor. While adding instances, helpful hints are displayed in the status bar (bottom left of the Schematic Editor window).

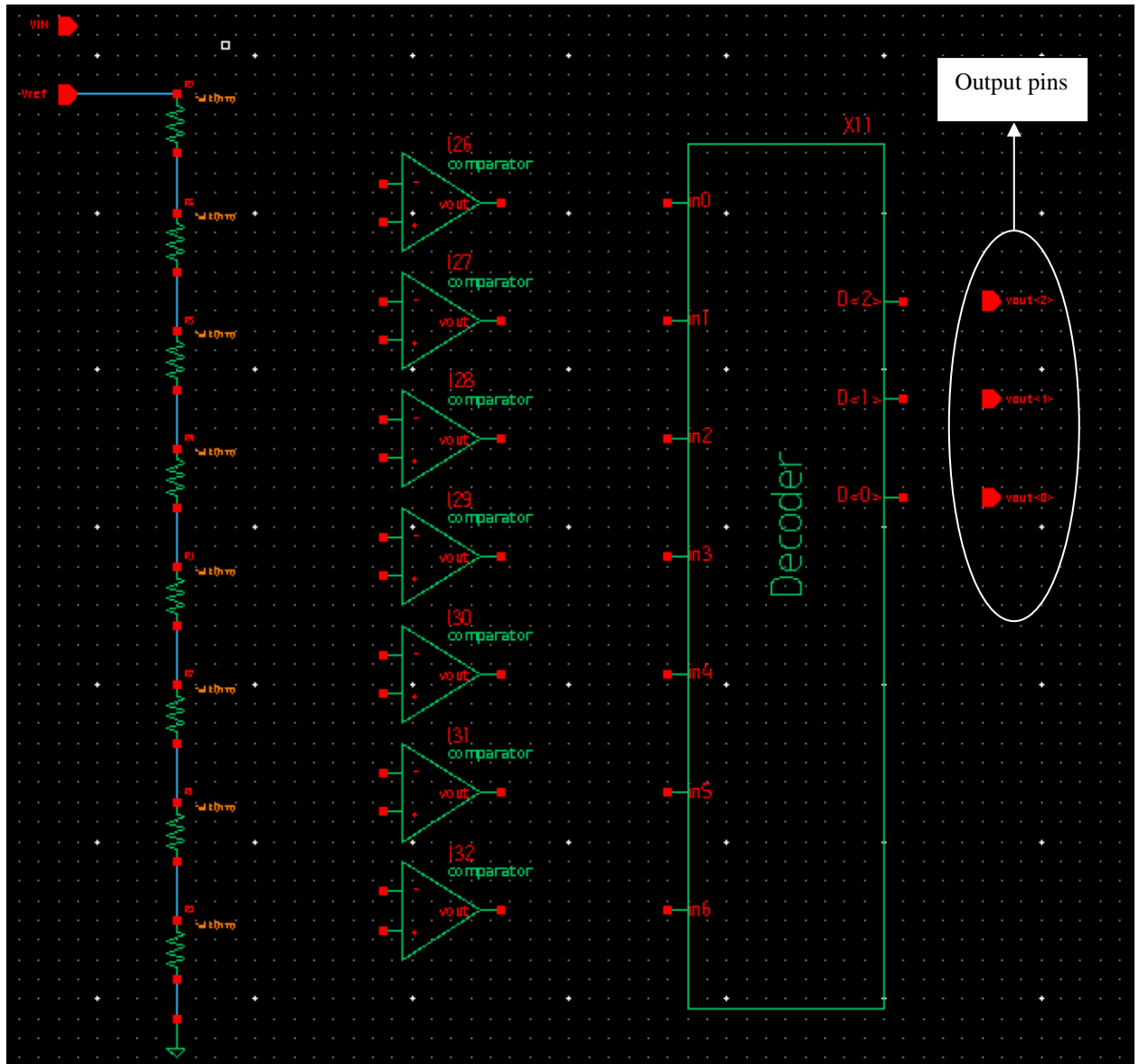
- Commit the second mouse click such that all instances are placed as shown.



53. After all the components are placed, the next step is to create pins to define the output signals of the circuit.

54. Place the output pin *vout*<0:2> using **Add → Pin** as shown below.

**Hint:** Enable the option **Expand** and **Array** and follow the messages in the status bar.



55. Now you have learnt two different techniques to create the design objects. You will learn more techniques in subsequent labs.
56. Save the schematic using **Design → Save**.
57. Close the *3bit-Flash* schematic window using **Design → Close**.

## ***Congratulations!***


You have successfully placed and edited the design objects in the schematic for creating the CMOS Differential Amplifier Circuit and have learnt useful techniques to create more complex designs.

## Task 27. Add Wires

---

After placing the components and pins to create the CMOS Differential Amplifier Circuit, the next step is to connect them according to the overall design functionality. To connect the components together in a schematic you use wires.

In this task, you will learn how to create the wires.

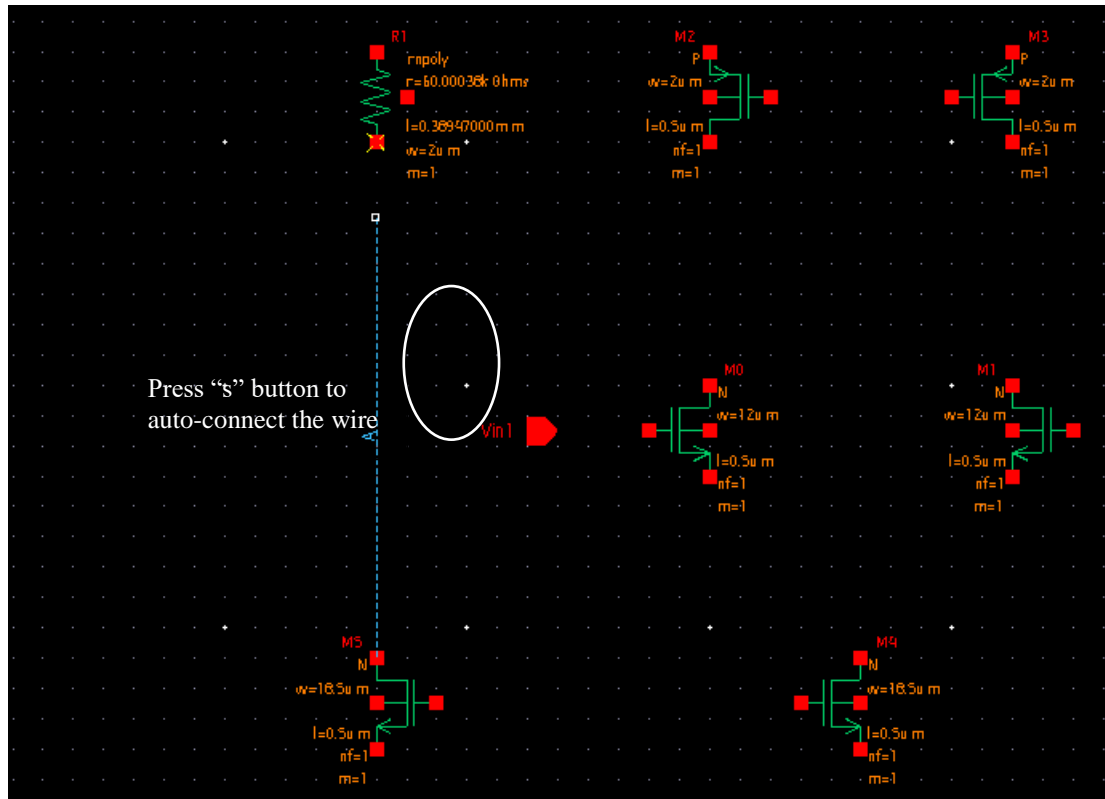
58. From the design library *Amplifier*, open a schematic cell *Diff* of the view name *schematic*.
59. Start creating the wiring by connecting *R1 Minus* pin to the drain pin of the *M5 nmos* transistor using **Add → Wire (w)** menu command or by clicking on Add Wire  button from the toolbar on the left side of the schematic window as shown.
60. During wire creation you have the capability to specify the created wire name in the opened status bar. In that case the created wire will have the name which is specified in the Nets section of Wire status bar. The below screenshot shows how to create a net name with A name. The name stacking and array naming options are supported during wire name creation.

**Note:** Every placed component has tiny red squares on its pins that you can connect to. During the wiring process, helpful hints are displayed in the status bar (bottom left of the Schematic Editor window).

61. Complete the wiring for the rest of the nodes as shown below using the same procedure, wire names will be assigned in the next task.

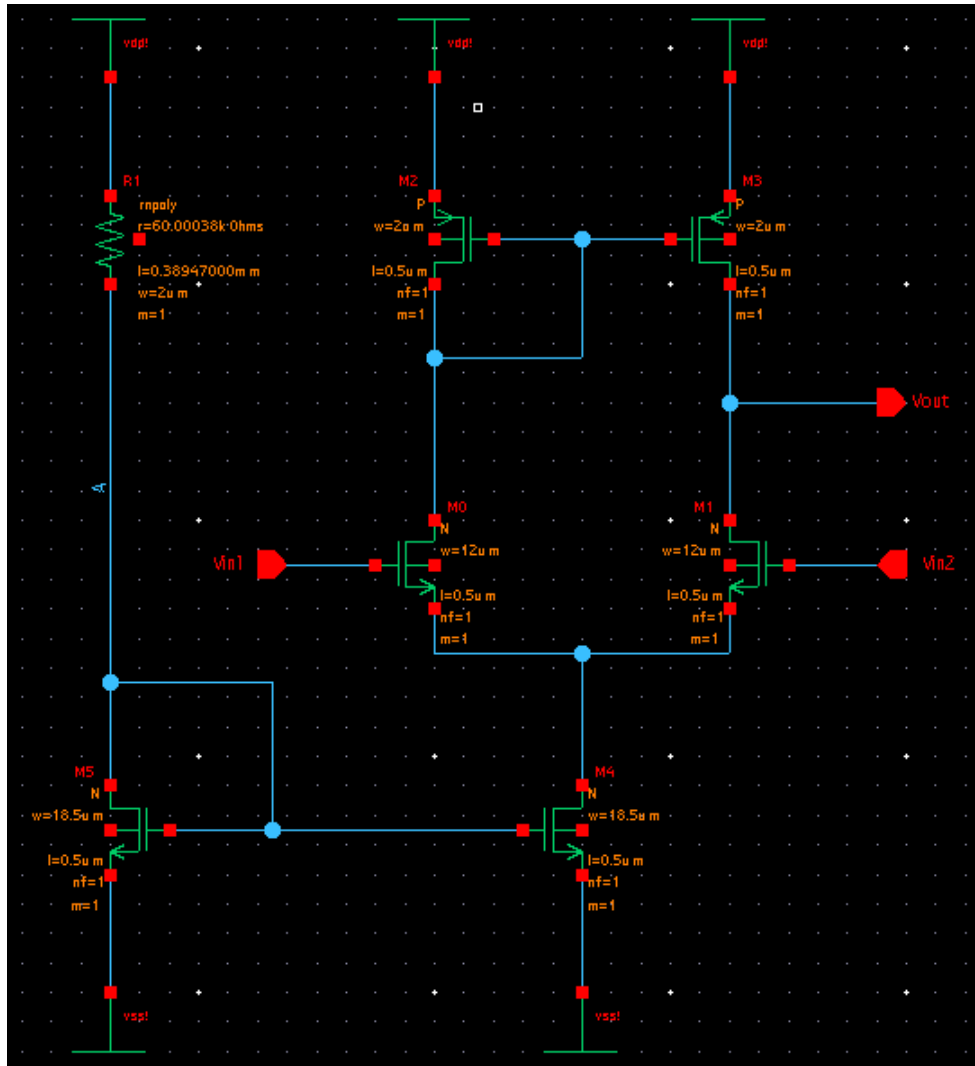
**Note:** Use the bind key **S** to snap the wiring to the auxiliary snap cursor, thus accelerating the wiring procedure.





**Question 6.** What happens to the wiring when the **Avoid Symbols** option is enabled?

62. At this point, the schematic should look like this:



## Task 28. Create Wire Name

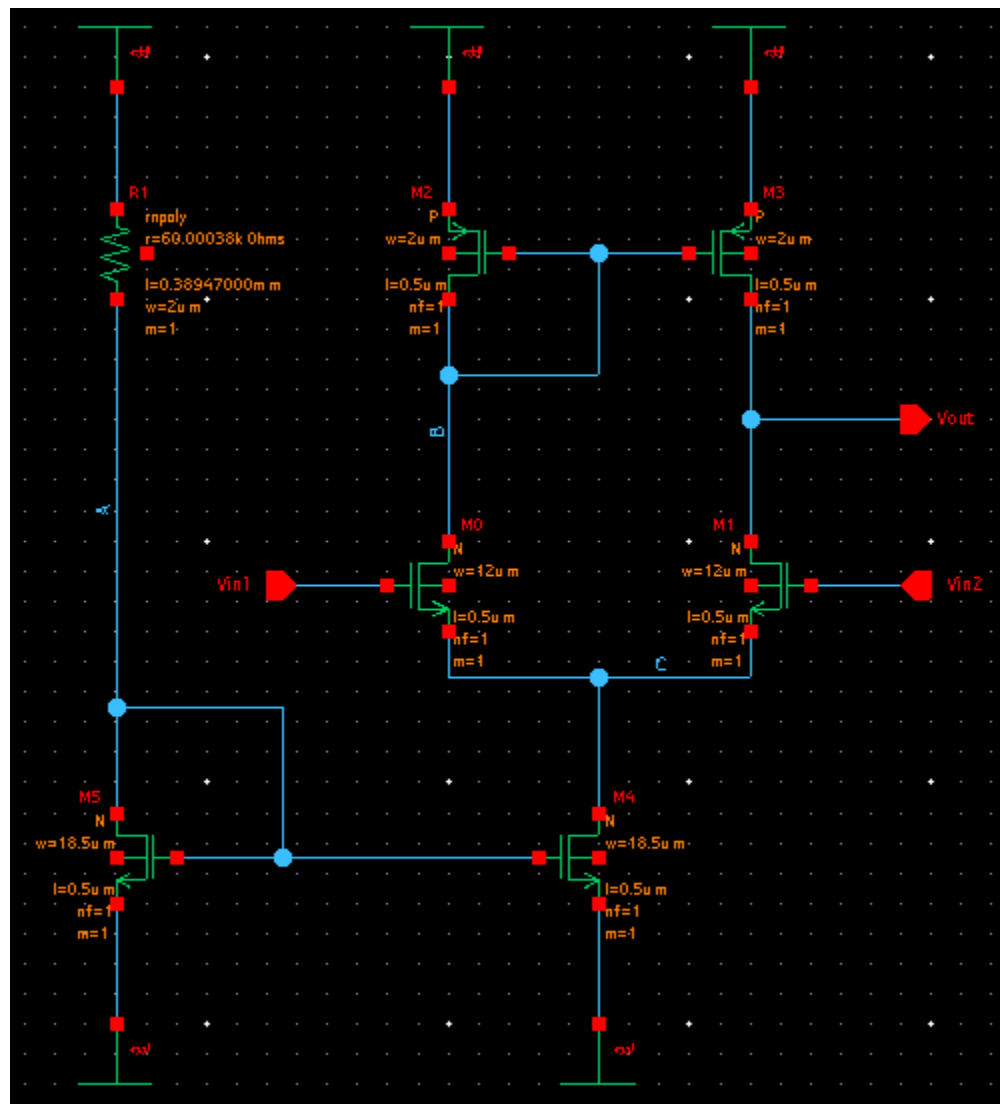
Whenever the wire is created (not started from a schematic pin) and if the wire name wasn't directly assigned during wire creation, then the tool automatically generates a net name for the wire. This net is called unnamed net and therefore, it should be named explicitly.

In this task, you will learn to name the wire as per your choice using **Add → Wire Name...**

63. Try to associate the following wire names in single command activation. (**Hint:** Use the concept of name stacking)
  - a. If the wire connecting the drain of M5 to the Resistor terminal has not name, which was assigned in previous task, associate the name A with that wire.

- Note:** There is a capability to specify the wire name in following format `<*"multiply">"WireName"`, in this case the wire will be created with the “wirename” name and “multiply” times, for example `< *4>a` will create “a” name 4 times.

**64.** At this point, the schematic should look like in the image below:



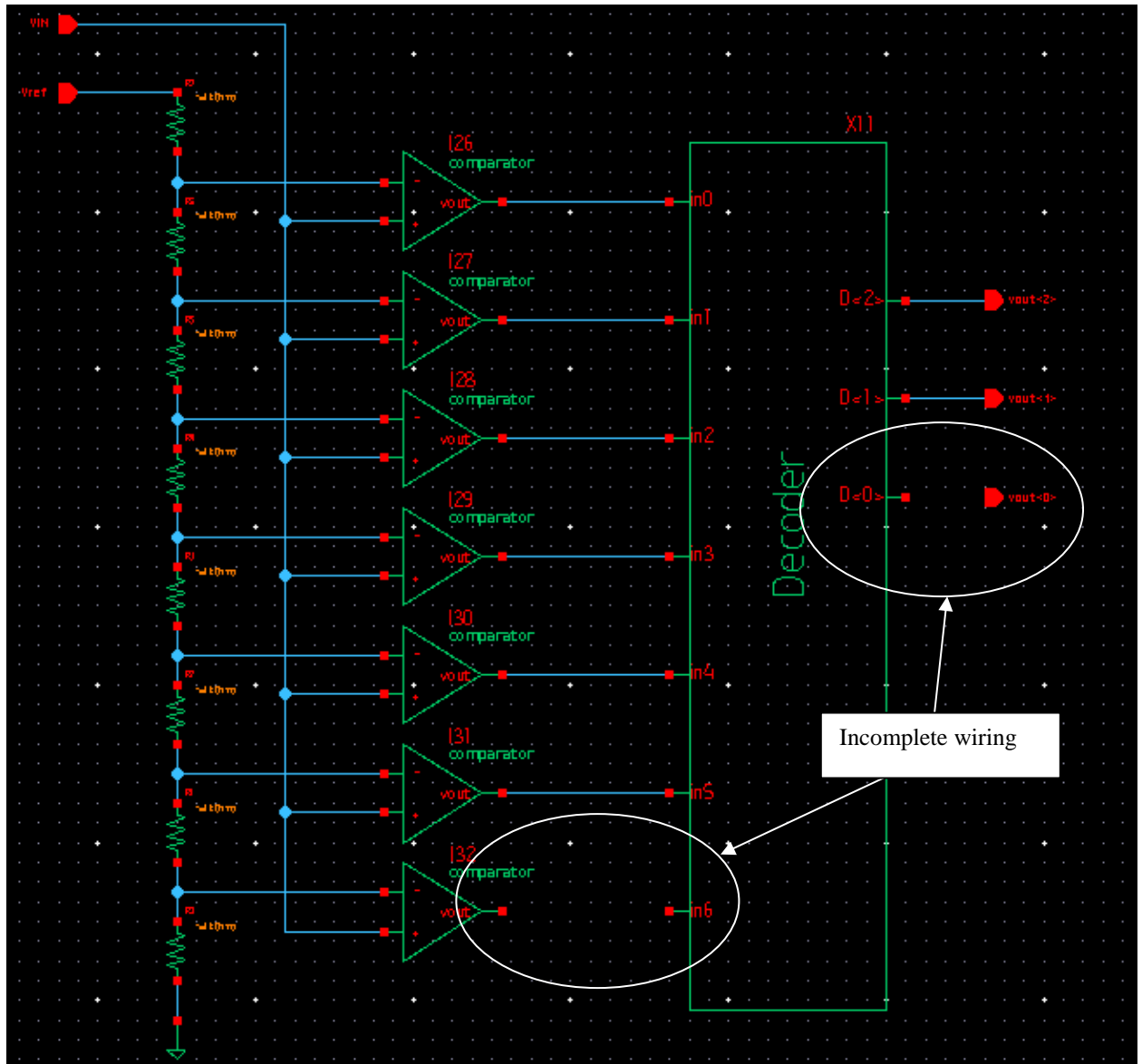
- Lab 1-51**  
Copyright © 2018 Synopsys, Inc. All rights reserved.

## Task 29. Useful Concepts – II

In this task you will learn useful techniques related to wiring and wire naming.

66. From the design library *ADC*, open a schematic cell *3bit-Flash* of the view name *schematic1*.

**Note:** The *3bit-Flash* cellView will be opened in a Schematic Editor window showing the *3bit-Flash* schematic.



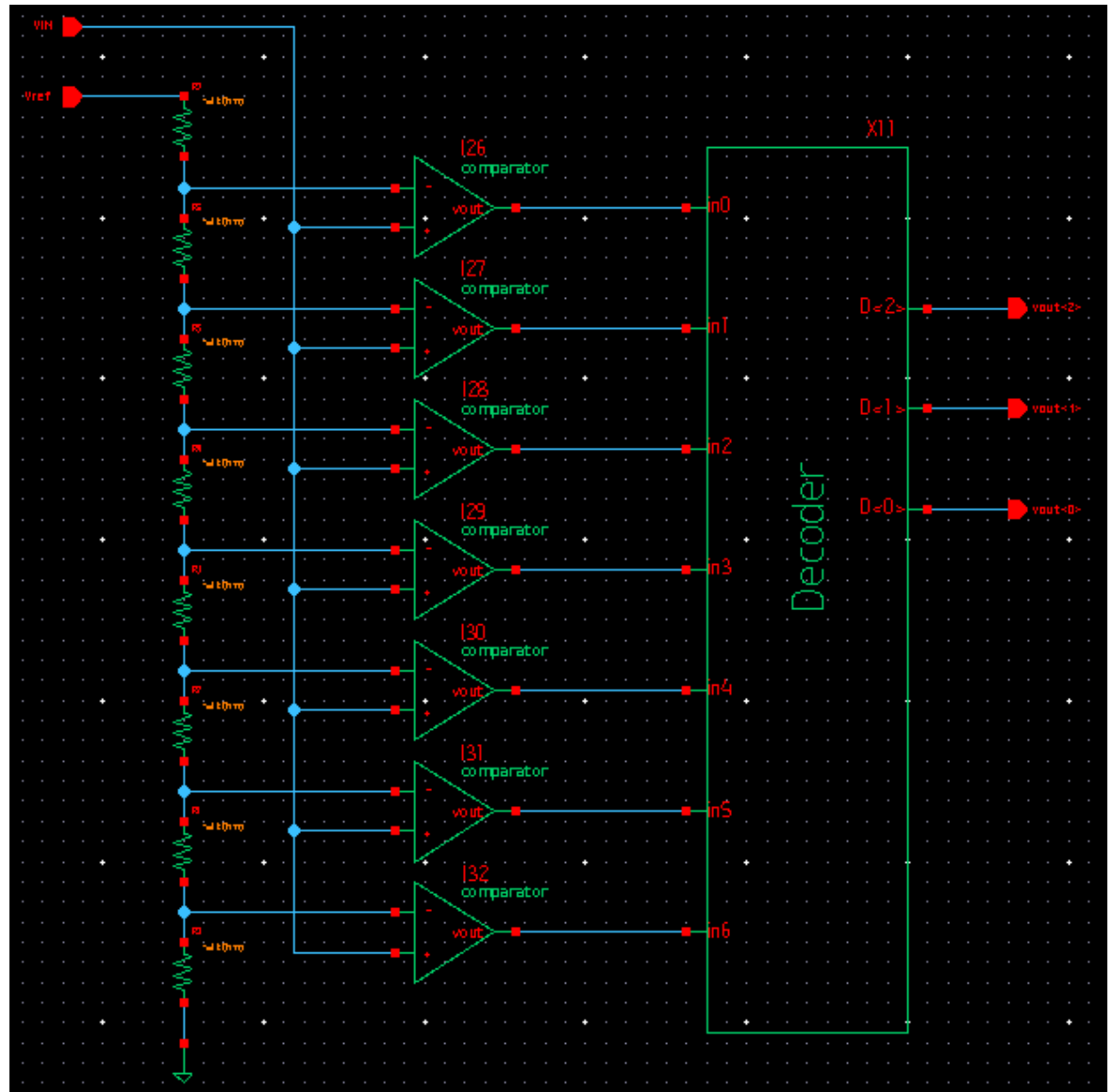
67. Here you will learn to use the technique of direct object manipulation for wiring. Place the mouse cursor over the *X32* instance pin *vout*.

**Note:** Make sure that only instance pin *vout* is active. If you are not able to visualize the pins properly, use zooming and panning from the **View** menu to set the viewport as per your choice.

68. Click and drag over the instance pin *in6* of the *Decoder*. When the instance pin *in6* becomes active, release the mouse button.

**Note:** As you start dragging the mouse, it starts wiring from the instance pin *vout*.

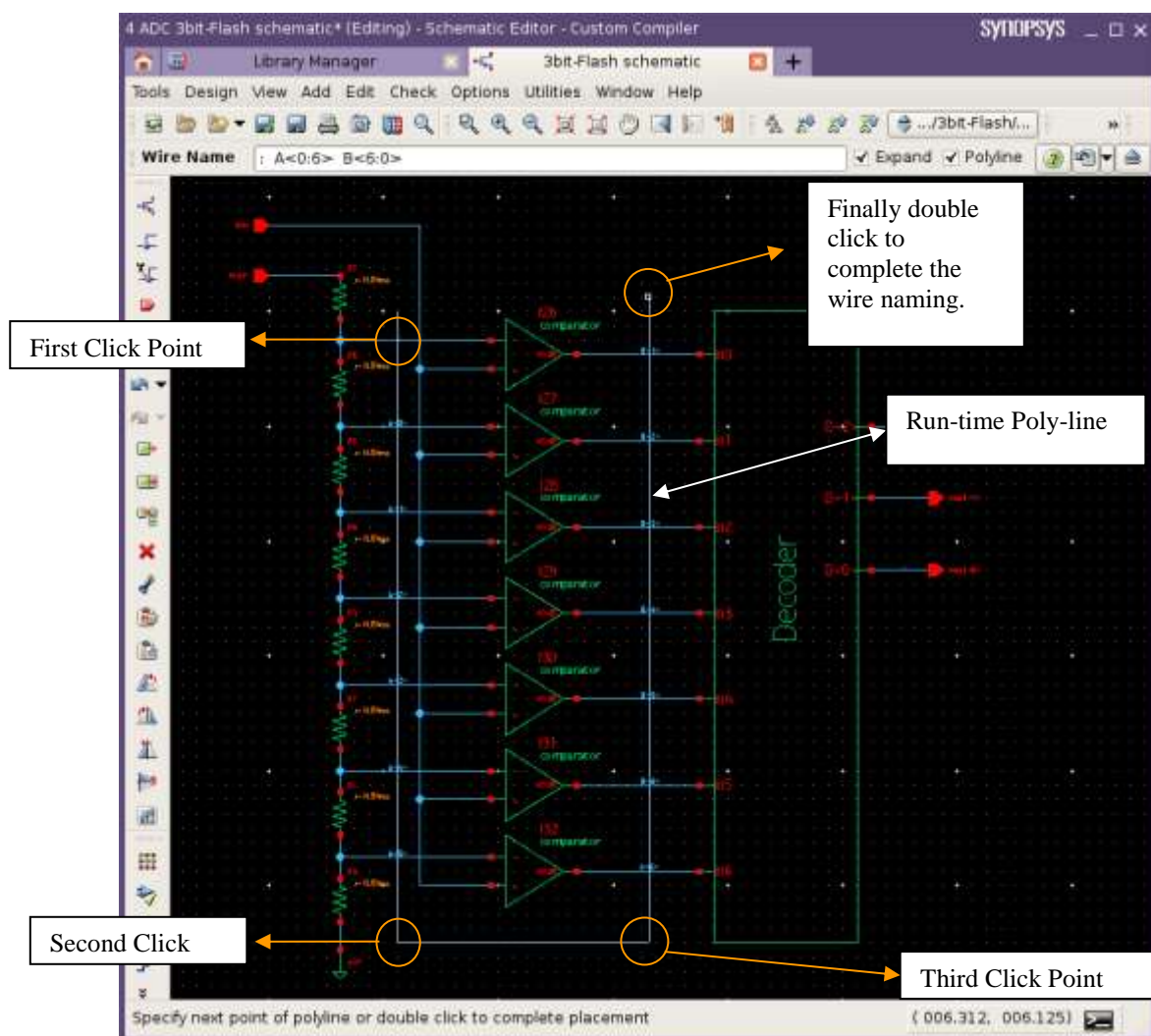
69. Similarly, route the net from instance pin *D<0>* to schematic pin *vout<0>*.
70. This completes the wiring of *3bit-Flash ADC*.



71. After wiring the next step is to name the unnamed nets/wires. You can see that you have seven unnamed nets which connect *comparator* instances to *decoder* instance and there are another seven unnamed nets which routes from resistor to - pin of the *comparator* instances.

If you have to associate the wire names to these 14 nets one by one, it's really time consuming. Here you will learn how you can name these nets in a more efficient way.

72. Try to associate the wire name to each of these 14 nets using **Add → Wire Name...** by using the following steps:
  - Select the **Expand** and **Polyline** options. Observe the status bar for the next instruction.
  - Enter the names as:  $A<0:6>$   $B<6:0>$  to name 14 nets.
  - Specify the first point just above the wire connecting *decoder* instance pin *in0*. Follow the figure and status bar instruction to complete the wire naming.



73. This completes the wire naming. Save the design.
74. Close the *3bit-Flash* schematic window using **Design → Close**.

## Congratulations!

You have successfully routed the design objects in the schematic for creating the CMOS Differential Amplifier Circuit and have learnt the small techniques to complete the task in a more efficient way.


### Task 30. Check and Save the Design

---

When capturing a design in a schematic it is important to verify the connectivity of the schematic before netlisting to avoid simulation debugging. It is also important to save your designs frequently so that your database is up-to-date.

In this task, you will learn to check and save your CMOS Differential Amplifier Circuit schematic and understand the various ERC/SRC error checks performed on the schematic.

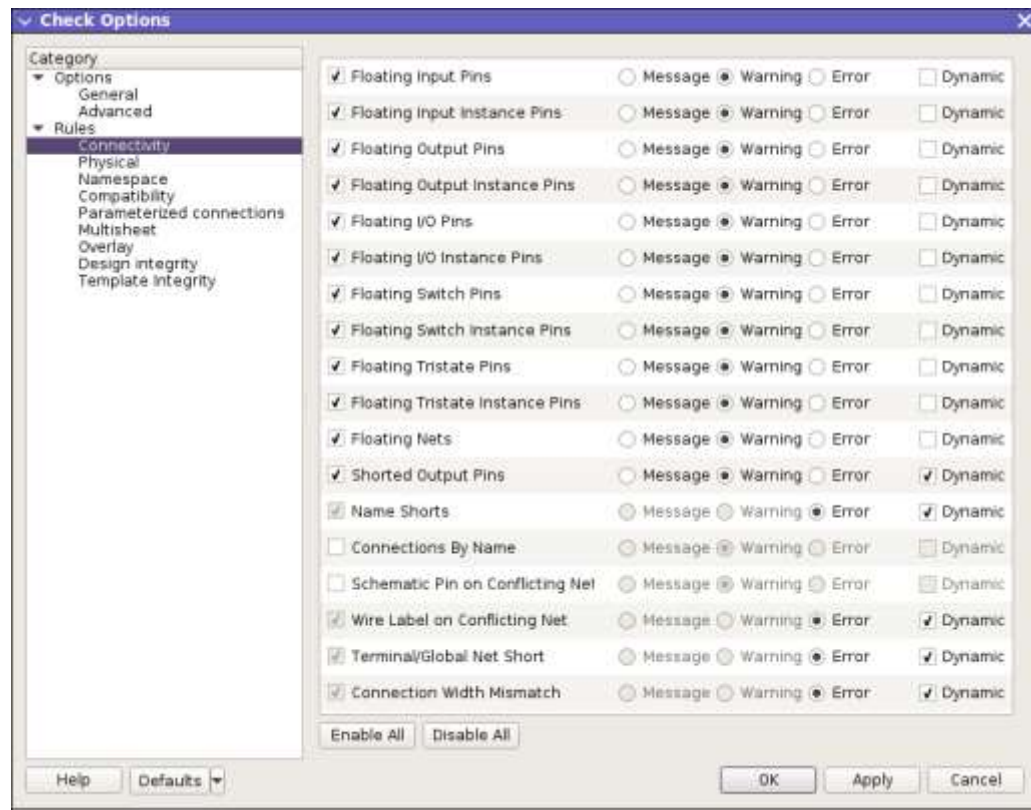
**75.** From the design library *Amplifier*, open a schematic cell *Diff* of the view name *schematic*.

**76.** Perform check and save operation using **Design → Check and Save** or by clicking on Check and Save  button.

**Note:** Observe the Console window to see the errors and warnings messages. Also observe the canvas to see if problem markers are generated.

**Question 8.** How many errors and warnings are generated in your check and save operation? What are these errors and warnings?

**77.** Open the ERC/SRC rule setup using **Check → Setup...** to see the various rules setup performed during check and save.



78. Set the severity level for rule **Floating I/O Instance Pins** to **Error** and click **OK** to commit the changes.
79. Perform the check on the schematic using **Check → Current View** menu command or by pressing the bind key **F7**. The following messages appear on the Console.

**Note:** It opens up the Marker Browser assistant docked on the right side of the schematic window showing all the violations.

It also brings up a dialog to save the design. Click **Save**.



**Question 9.** Can you identify the difference between Step 2 and Step 5?

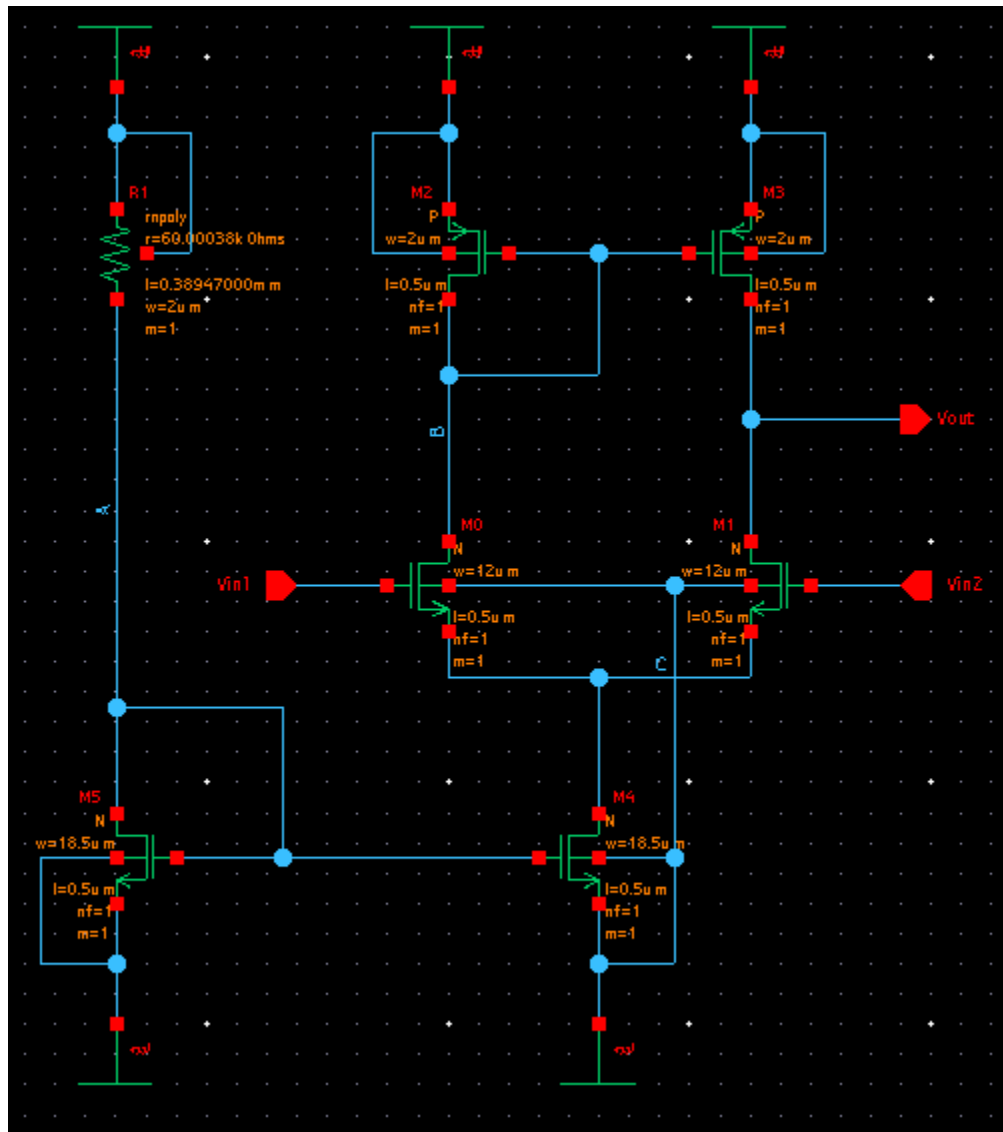
80. Connect the bulk of all MOSFETs as shown below and perform check and save operation again.



**Note:** If you have made the bulk connections correctly, all the violations will disappear in the Marker Browser.

**Question 10.** What message did you get after fixing the above errors and warnings?

81. The final schematic should look like this:



82. You have successfully verified your circuit. Close the *Diff* schematic window.

***Congratulations!***

You have successfully created the schematic of the CMOS Differential Amplifier Circuit!

# Answers / Solutions

## Task 10. Create and Place Instances

**Question 5.** How can multiple instance names be entered in the **Add→Instance** form?

Multiple instance names can be entered by separating the names with a space in the **Name** field. This is called Name Stacking. When a stack of names exists, the first name from the left will be processed first.

## Task 11. Edit Instance Parameters

**Question 6.** How are the default values of the parameter displayed in the Property Editor assistant?

83. If the value of the parameter is the default value then the prompt of the parameter are displayed in *Italics*.

## Task 9. Add Wires

**Question 7.** What happens to the wiring when **Avoid Symbols** option is enabled?

84. The tool does not allow the wires to be routed over the symbols when this option is enabled.

## Task 10. Create Wire Name

**Question 8.** How do you associate the wire name if you click on the open space instead of selecting wire?

85. If the defined point is in open canvas you will be required to select the wire segment by the second click to associate the wire name.
86. Once the first point is defined in open space, the label will be displayed at that location and a ghost line will be displayed from the label origin to the cursor, and you will be prompted to select a wire segment. Once you click on a wire segment, the new label will be grouped with that segment and set as the child of the group (the wire segment is the parent), and the ghost line will be removed. If you click anywhere other than on a wire segment, this click will be ignored.

## Task 12. Check and Save the Design

**Question 9.** How many errors and warnings are generated in your check and save operation? What are these errors and warnings?

- 87. Ideally, there should be 7 warnings, if you have done each step successfully.
- 88. These warnings are associated to the floating bulk terminals of the mosfets which are un-connected. Error markers of golden rectangle get created on each of these floating terminals.

**Question 10.** Can you identify the difference between Step 2 and Step 5?

- 89. The difference between Step 2 and Step 5 is about the severity level of the rule and also the color of the generated markers on each of the floating terminals.

In Step 2 severity level of the rule is **Warning** and the marker generated are golden rectangles whereas in Step 5 severity level of the rule is **Error** and the markers generated are white rectangles.

This clearly shows how the markers can be differentiated on the canvas depending on their severity.

**Question 11.** What message did you get after fixing the above errors and warnings?

- 90. If there is no violation in your schematic, you will get the following message in your Console window.

*No rule violations found in "Amplifier/Diff/schematic"*

# 4

## 4. Schematic Entry Using Wiring Accelerators

### Learning Objectives

The main objective of this lab is to familiarize you with the new features of Custom Compiler Schematic Environment.

After completing this lab, you should be able to:

- Use Block Creation functionality
- Align design elements using Align command
- Routing easily multiple pins with River Route
- Create and label multiple wire stubs at once



**Lab Duration:**  
30 minutes

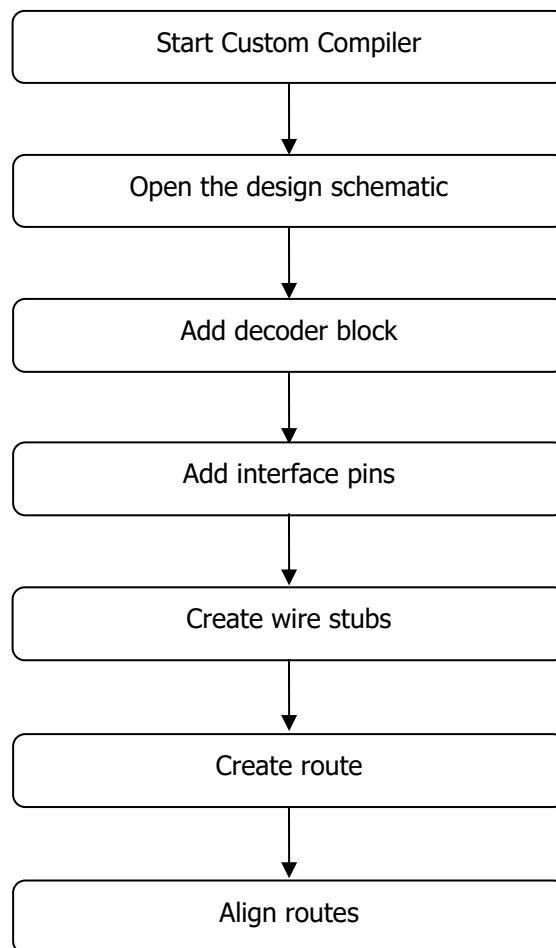
# Introduction

The goal of this lab is to design a 3 bit Flash ADC circuit using advanced capabilities of Custom Compiler Schematic Editor. Follow the instructions in the tasks below to accomplish this goal, but make sure you try or play a little with the commands in order to increase the probability of generating questions while you are in the class and deepen your understanding of the tool.

In this lab, you are provided with *lib.defs* file which contains the mapping of *reference40nm* (Reference design library), *analogLib*, *basic*, *sample* and *sheets* libraries (libraries shipped with the installation). These libraries contain the primitive components (passive and active elements) and other design components to build the design. You will use these libraries as reference libraries to build your schematics.

# Flow Overview

## Lab1 Tasks



# File Locations

All files for this lab are located in the directory  
*SE\_Advanced\_Schematic\_Editing\_Lab1*.

## Directory Structure

|                                    |                           |
|------------------------------------|---------------------------|
| SE_Advanced_Schematic_Editing_Lab1 | Current working directory |
| ADC                                | OpenAccess Design library |
| Amplifire                          | OpenAccess Design library |
| lib.defs                           | Library definitions file  |



# Instructions

## Task 31. Start Custom Compiler

---

91. In the Unix terminal window, change your current working directory to *SE\_Advanced\_Schematic\_Editing\_Lab1*. This will be your working directory for this lab.
92. Start Custom Compiler from the Unix prompt.

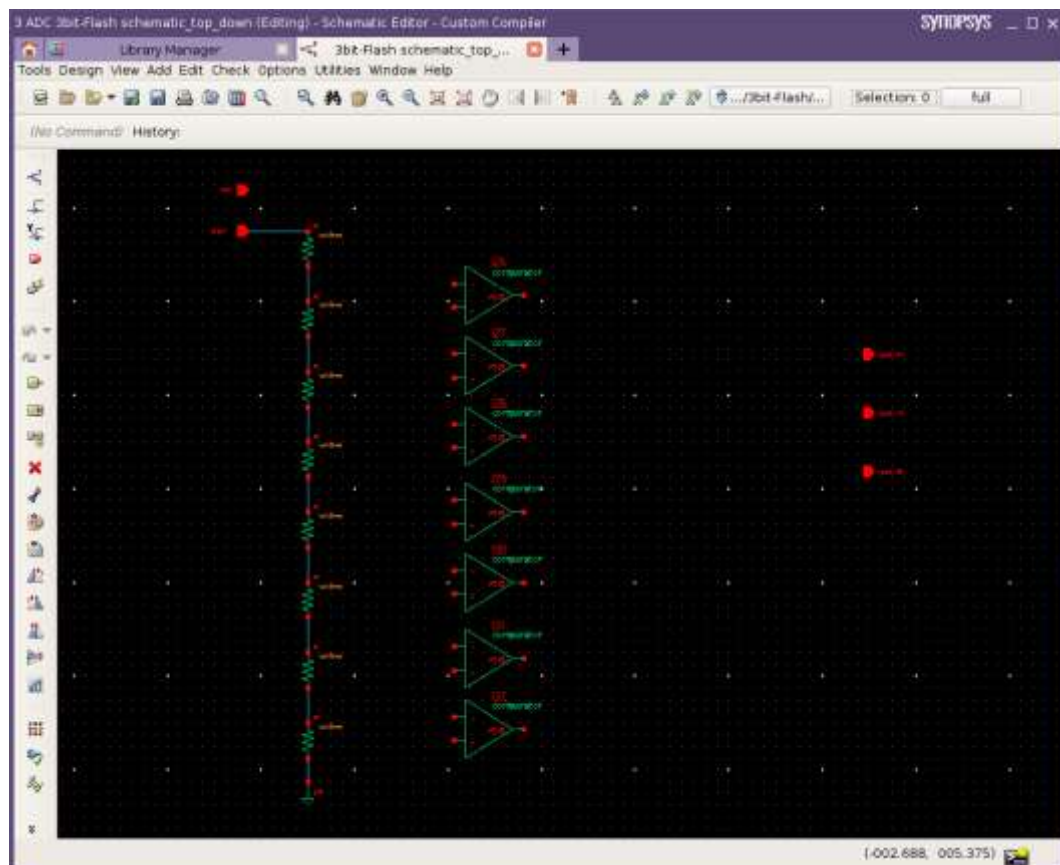
```
custom_compiler &
```

## Task 32. Open the Design Schematic

---

1. From the design library *ADC* open the *schematic\_top\_down* view of *3bit-Flash* cell.

**Note:** Analog to digital converter ADC cellView will be opened in a Schematic Editor window. The circuit is used to convert analog signals to digital.




You can see decoder missing in schematic. Let's add it.

### Task 33. Add decoder block

---

In this task you will add decoder block, using block creation functionality.

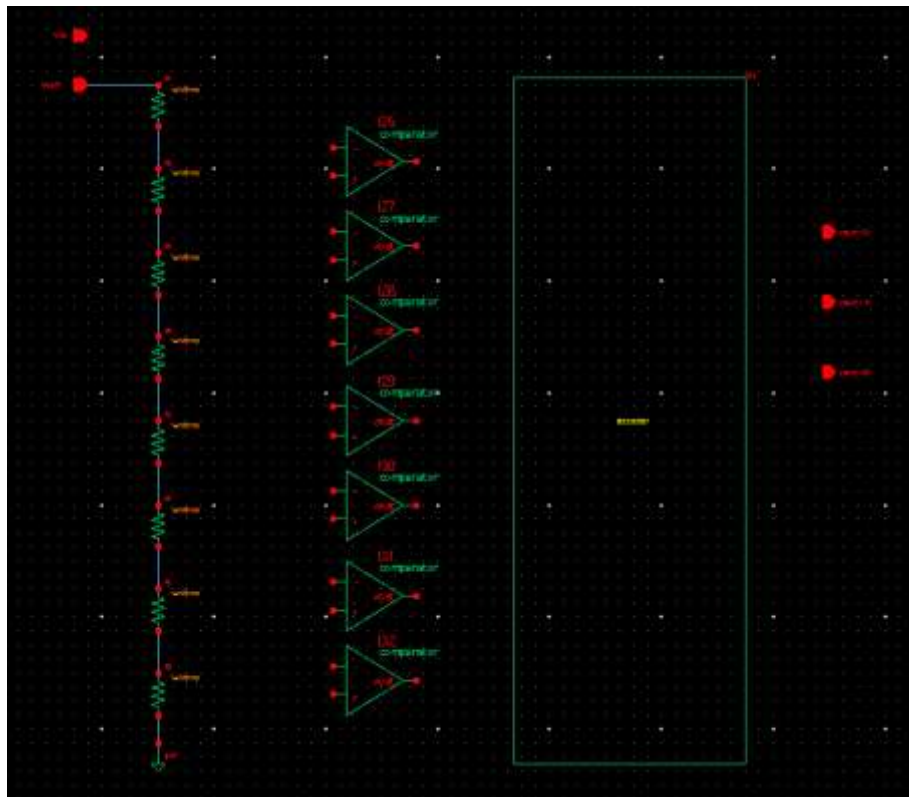
1. Invoke **Add->Block**  Block... command and enter *decoder* in the *Cell* field in COT of the command.

**Note:** This will create new symbol view in ADC library called *decoder*.

COT of the command is shown in the picture below:




2. Click on canvas to start drawing the symbol. Click again to finish symbol creation.
3. Terminate the Add -> Block command using Esc key. At this point schematic should look like this:



Now let's see how we can connect outputs of comparators to the decoder.

## Task 34. Add interface pins

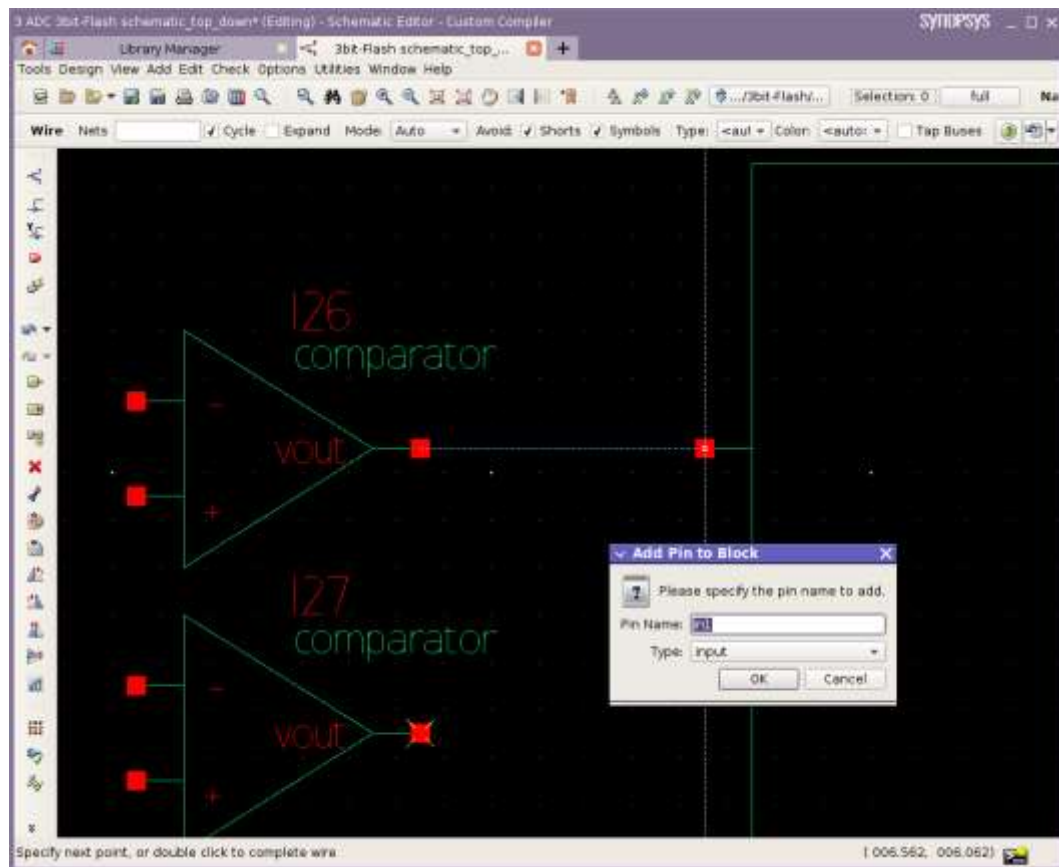
In this task you will create interface pins for *decoder* block.

1. Invoke **Add->Wire (W)**  **Wire** command and start routing wire from the comparator output to the selection edge of the decoder.

**Note:** You can see pin for decoder symbol is dynamically created.

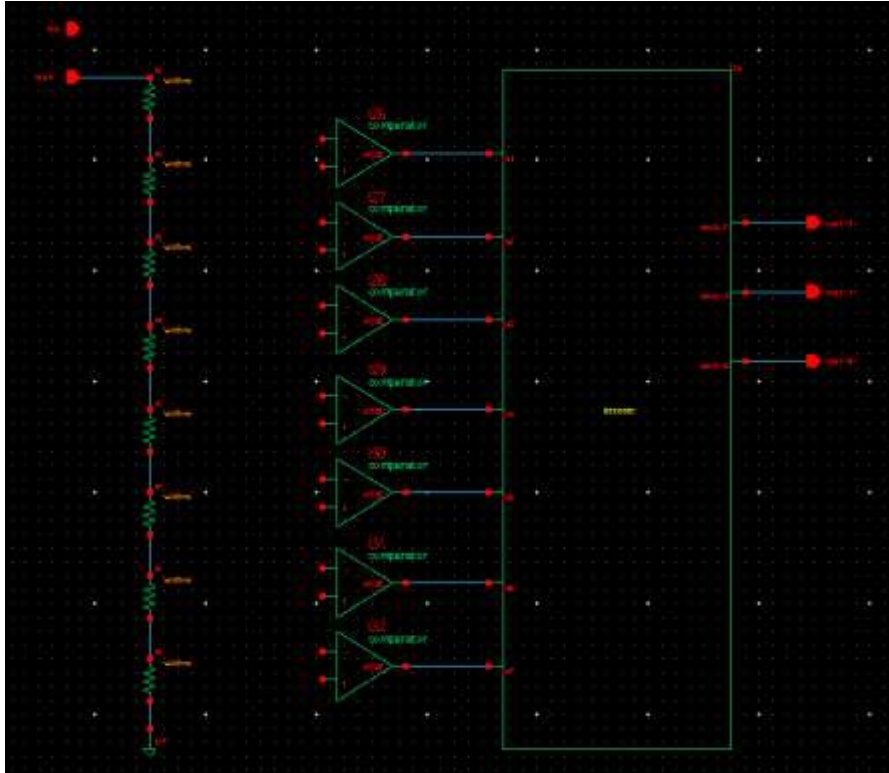
2. Commit the wiring.
3. Enter pin name and pin type.

**Note:** By default pins created in left side of the symbol are input, right side are output, top and bottom pins are type of inputOutput.



4. Connect all comparator outputs to the decoder.
5. Connect vout<2>, vout<1> and vout<0> pins to decoder.

Schematic should look like this:



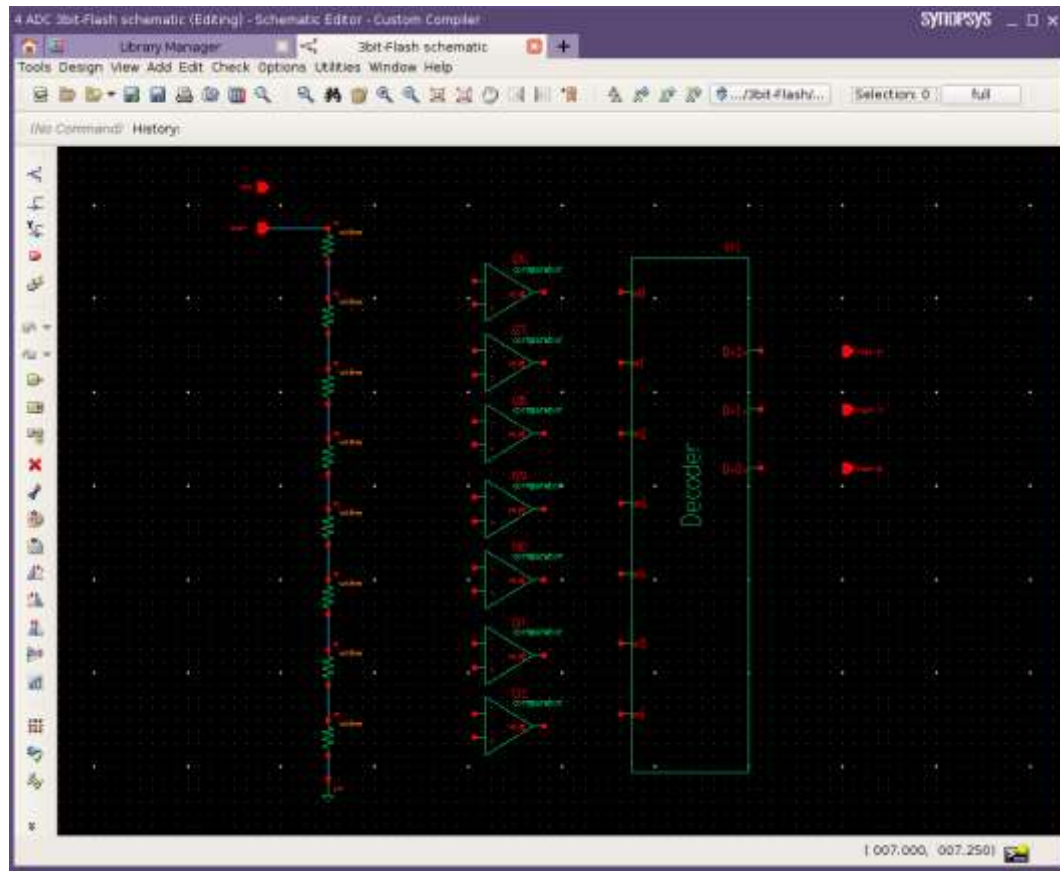
## Task 35. Create wire stubs

---

In this task you will use Add Stub feature to easily create named wire stubs.

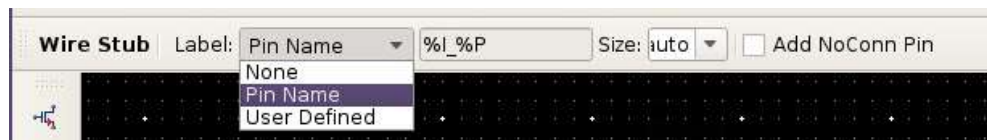
1. Open *schematic* view of *3bit-Flash* cell from *ADC* library. Schematic should look like this:

**Note:** *Schematic view of 3bit-Flash cell will be opened in a Schematic Editor window.*



2. Invoke **Add->Stub**  **Stub** command.

COT of the command is shown in the picture below:



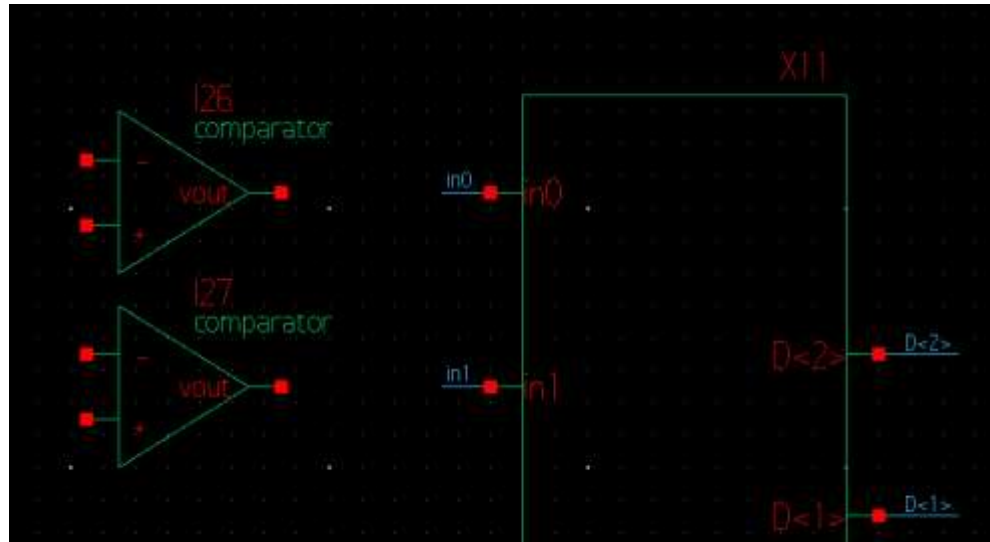
3. From Label drop down menu select *Pin Name* and from Size select *10*.

**Note:** This will create labels, same as instance pin name, for created stubs and will make sure that the length of the stubs will fit the longest label name. You can explicitly specify stub length by modifying *Size* field on COT. If *Label* is selected as *User Defined*, it is possible to specify wire stub naming template, using %I and %P placeholders for instance and terminal names. *Add NoConn Pin* will add NoConn pin symbol to the created stubs.

4. Place mouse over the Decoder instance.

**Note:** Preview for wire stubs and labels will be shown. Move mouse close to right side of symbol and stubs will be shown only for pins located at right side.

5. Move mouse to left hand side of decoder cell and commit stub creation either by hitting Enter key or pressing mouse left button.





## Task 36. Create route

---

In this task you will route multiple pins to wire stubs created in previous task.

1. Invoke **Add->Route**  **Route** command.

**Note:** This command can be used to create route either for pin-to-pin or pin to wire.

2. Switch to partial selection mode  .

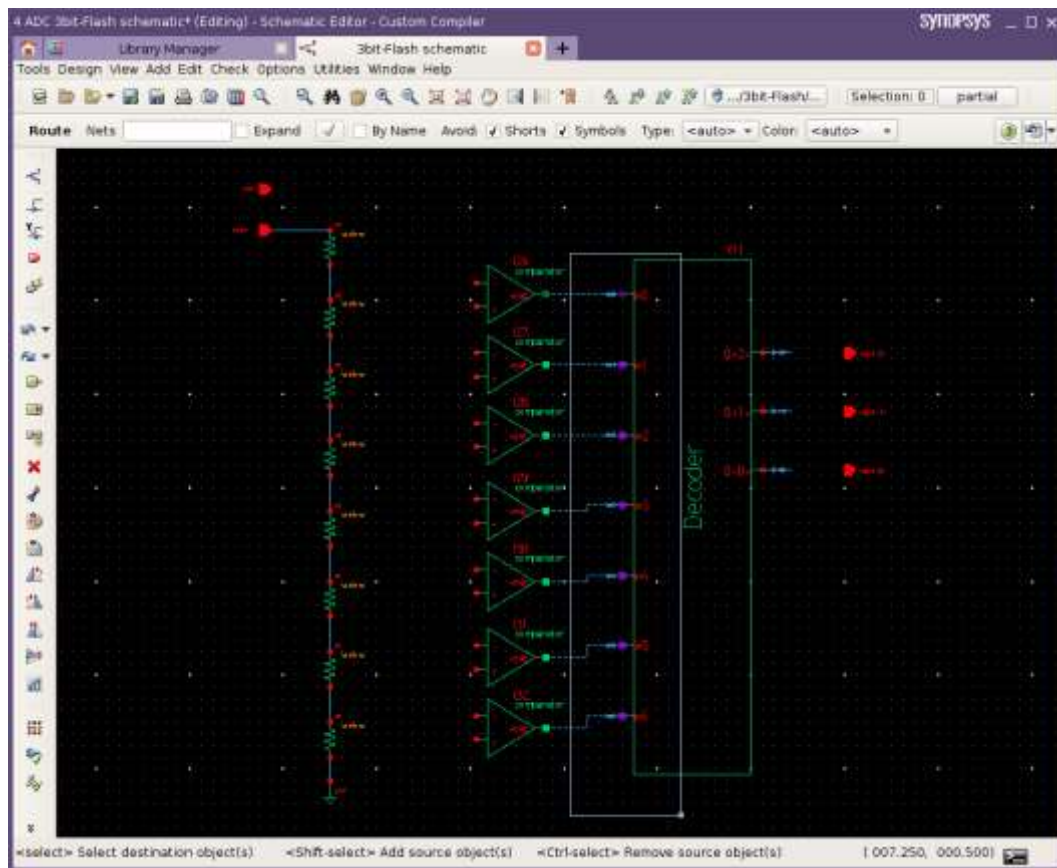
3. Select all output terminals of comparators. They will be added to the source pins collection.


**Note:** You can use Shift-Click to add other pins to selection or Ctrl-Click to remove from it.

4. Select wire stubs connected to decoder inputs created in previous task.

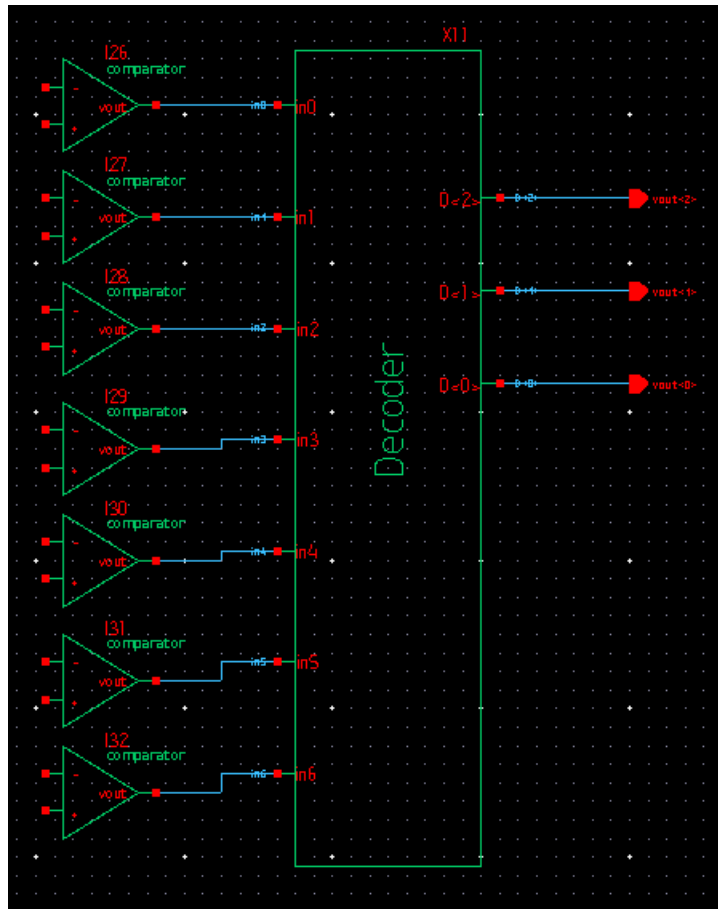
**Note:** Actual routing preview from pin to pin will appear.





5. Commit the routing using  button.
6. Use this command to route output terminals of Decoder to output pins vout<2>, vout<1>, and vout<0>.

After routing schematic should look like this:



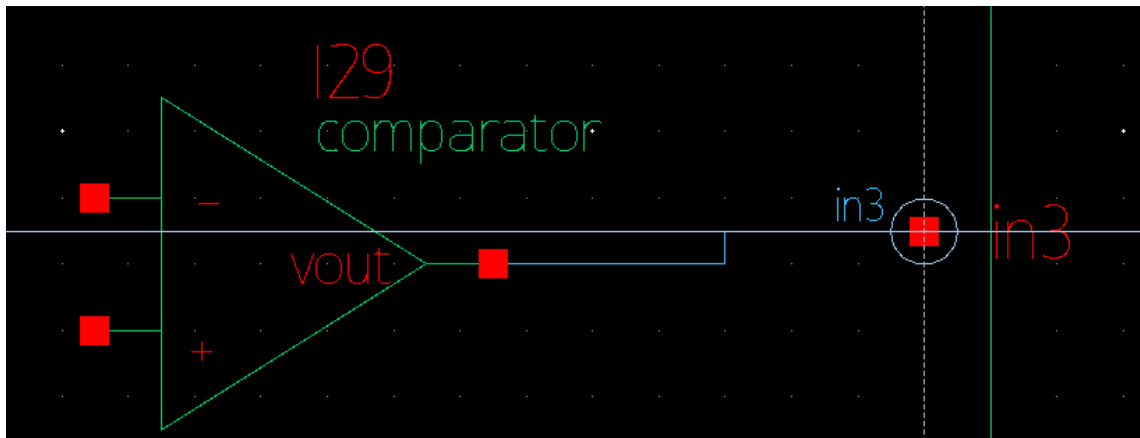
## Task 37. Align routes

In this task you will use Custom Compiler align feature to align non straight routings created in previous task.

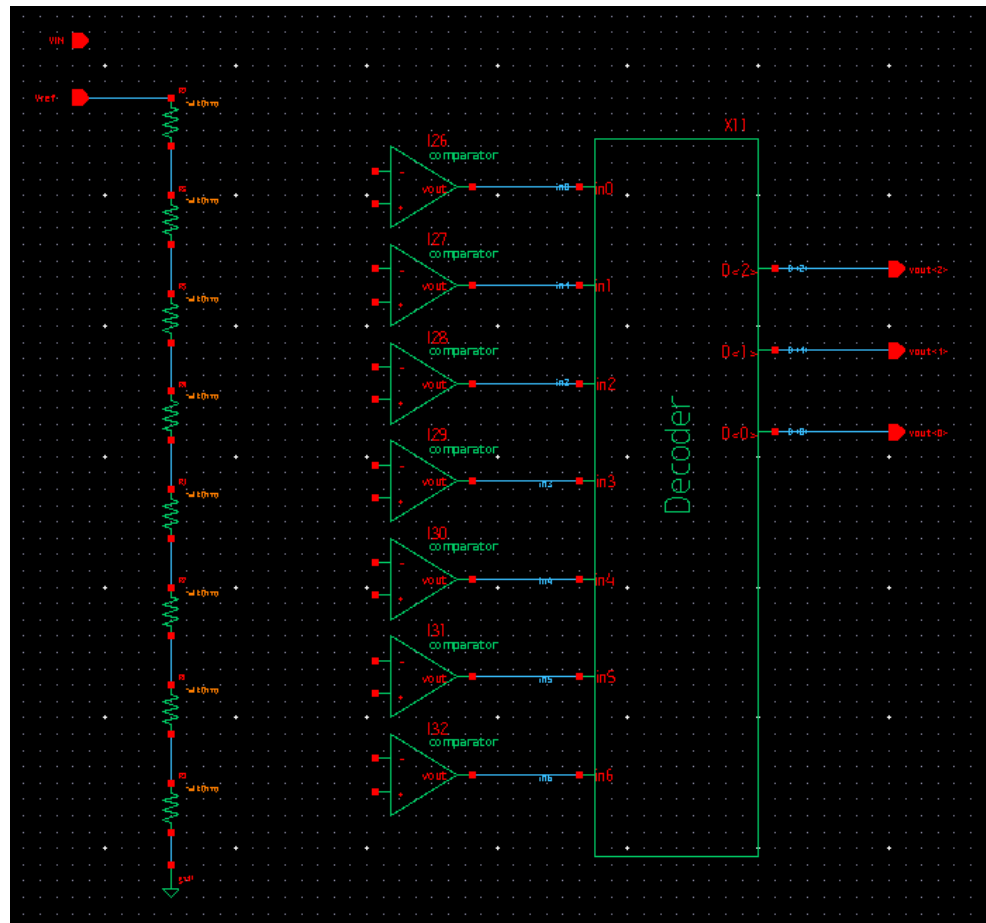
1. Invoke **Edit->Align** command or hit 4 bindkey.
2. Select alignment direction to be Horizontal.
3. Click on the input terminal on Decoder block, notice a bubble will appear at the source and a horizontal line is displayed. Now click on the output terminal of corresponding comparator.

**Note:** Note that comparator moves so pins are being aligned and you get a straight routing.





After alignment schematic should look like this:



*Congratulations!!!*

You have learned to use schematic entry and editing new features in Custom Compiler.

# 5

## 5. Symbol Creation

### Learning Objectives

The purpose of this lab is to allow you to learn the symbol generation features in the Custom Compiler Environment.

After completing this lab, you should be able to:

- Create the symbol
- Edit the symbol
- Add Selection Shape
- Check the symbol
- Save the symbol



**Lab Duration:**  
30 minutes

# Introduction

A symbol is a graphical representation of a hierarchical design. It has properties that describe characteristics of a model or schematic it represents. Symbols are useful when creating designs where it is impractical to show every transistor on the top level schematic (that is, hierarchical designs) and also very useful when you want to represent the design with language modules that do not have any schematic representation. During this lab, you will create a symbol of a CMOS Differential Amplifier schematic using the Custom Compiler Symbol generation flow.

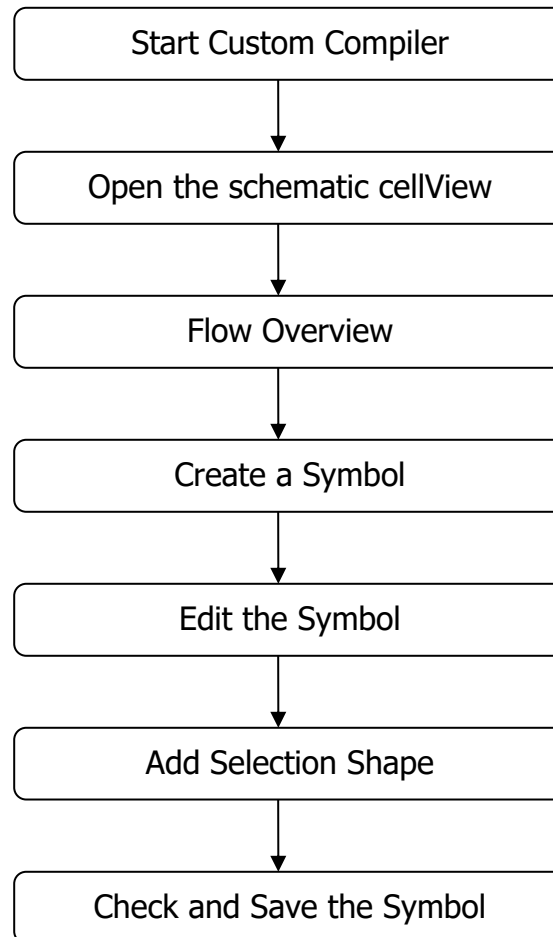
In this lab, you are provided with *Amplifier* OpenAccess database library and a *lib.defs* file. The *Amplifier* library contains the design of a CMOS Differential Amplifier circuit. You will use this schematic circuit to create your symbol.

In order to create the symbol of the Differential Amplifier schematic you need to understand the basic usage of the commands that are required to create it.

Please see the section “Data Creation”, “Editing” and “Symbol Generation” of the *SE command reference manual* for the commands that are used in this lab.

# Flow Overview

## Lab 1 Tasks



# File Locations

All files for this lab are located in the directory *SE\_Symbol\_Creation\_Lab1*.

## Directory Structure

|                                      |                           |
|--------------------------------------|---------------------------|
| <code>SE_Symbol_Creation_Lab1</code> | Current working directory |
| <code>Amplifier</code>               | OpenAccess Design library |

## Relevant Files

|                       |                          |
|-----------------------|--------------------------|
| <code>lib.defs</code> | Library definitions file |
|-----------------------|--------------------------|

## Answers & Solutions

There is an *ANSWERS / SOLUTIONS* section at the end of this lab. You are **encouraged** to refer to this section often to verify your answers, or to obtain help with the execution of some steps.

# Instructions

## Task 38. Start Custom Compiler

---

93. In the Unix terminal window change your current working directory to *SE\_Symbol\_Creation\_Lab1*. This will be your working directory for this lab.
94. Start Custom Compiler from the Unix prompt.

```
custom_compiler &
```

## Task 39. Open the schematic cellView

---

95. From the library *Amplifier*, open a schematic cell *Diff* of the view name *schematic*.

**Note:** The *Diff* cellView will be opened in a Schematic Editor window showing the Differential Amplifier schematic.

In order to make the view port of the schematic as per your choice you can always use zoom in and zoom out using **View → Zoom → Zoom In (Ctrl+Z)** or **Zoom Out (Shift+Z)**.

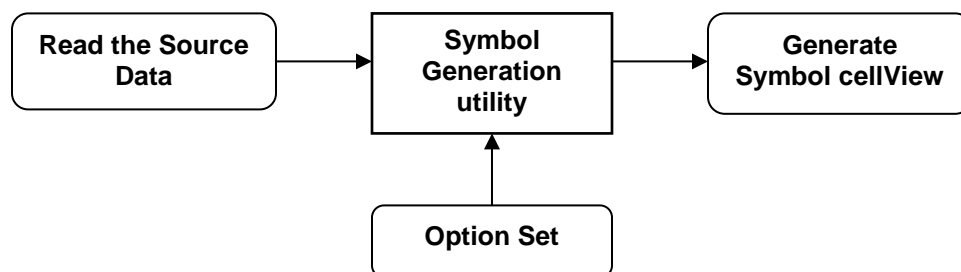
## Task 40. Symbol Generation Flow Overview

---

In Custom Compiler, the schematic symbol can be automatically generated in two ways:

- From the cellView
- From the Pinlist

This figure shows the simplified flow of the Symbol Generation within Custom Compiler.



The Symbol Generation from cellView involves two processes; first it reads the pin and parameter information from the specified source cellView and then takes the user inputs and symbol settings (**Option Set**) to generate the final symbol cellView.

**Note:** The **Option Set** contains list of symbol data (symbol construction parameters - SCP) available for user control.

Here the symbol data refers to the pin spacing, size of the pins, stub length, size of the rectangle, position of the labels and annotations in the symbol.

The Symbol Generation from Pinlist reads only the pin information either from the source cellView or from the user defined list. It then takes the symbol settings (**Option Set**) to generate the final symbol cellView.

You will create a symbol of the Differential Amplifier schematic using the above flows. The guidelines to achieve this task will be provided in the subsequent tasks.

## **Task 41. Create a Symbol**

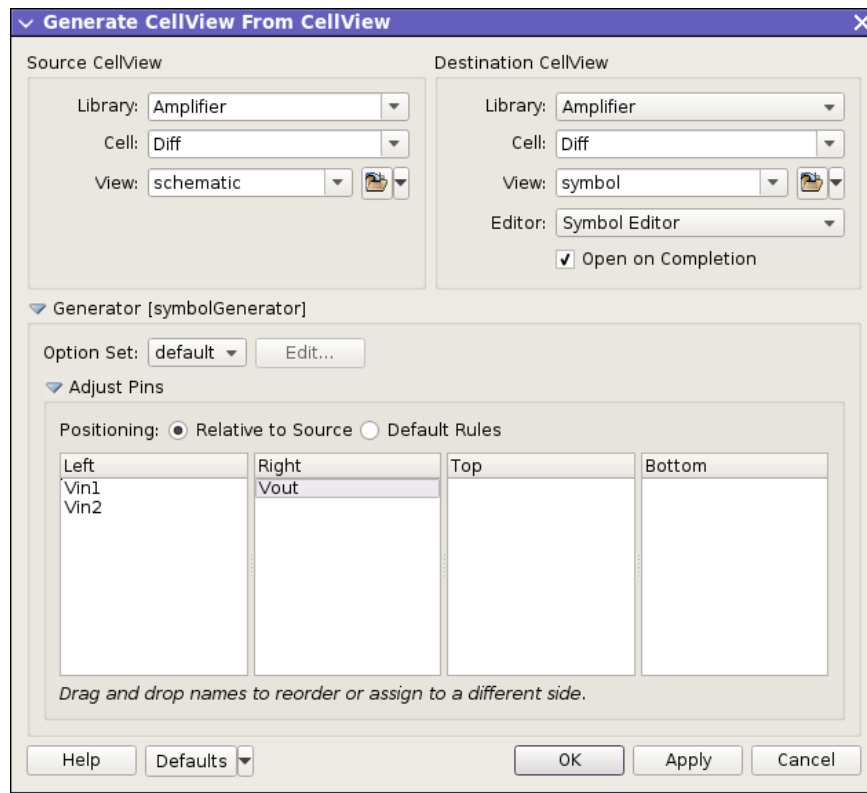
---

Once the schematic design is done, the next step is to create a symbol. Symbols are used in creating hierarchical schematics or creating test benches for simulation.

In this task, you will learn to create the schematic symbol of the CMOS Differential Amplifier schematic.

96. Create a symbol cellView with the default settings using **Design → New CellView → From CellView ... (Y)** such that the input pins are on the left side and the output pins are on the right side, as it is shown in the below picture:

**Hint:** Use **Adjust Pins** option to modify the pin positions.



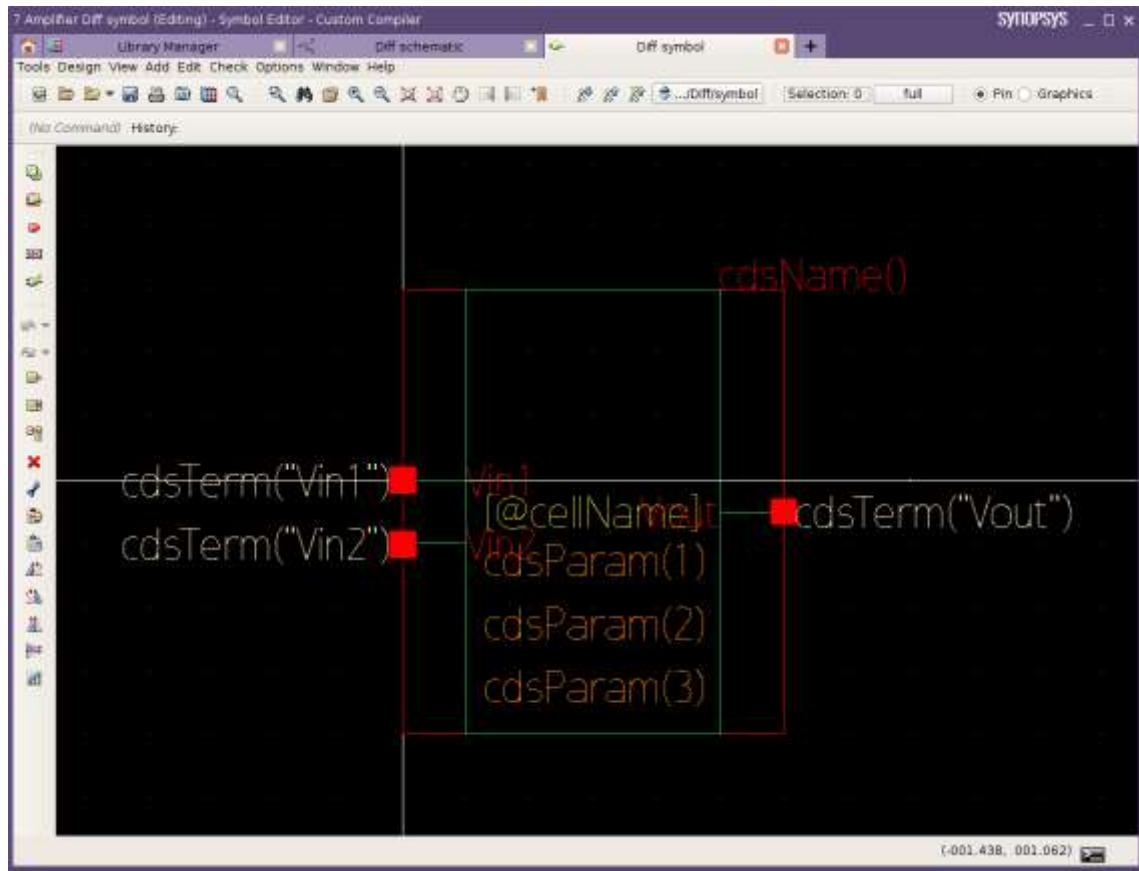
**Note:** Observe that the tool takes the current edit cellView as the default source. In this case it is *Diff* cell from the library *Amplifier* with the view name *schematic*.

Executing the form will open up the *Diff* symbol cellView in a Symbol Editor window, showing your new symbol. This is the symbol representation of your schematic.

**Question 11.** How many pins are created in the symbol cellView and what are their names?

**97.** At this point, the symbol should look like this.



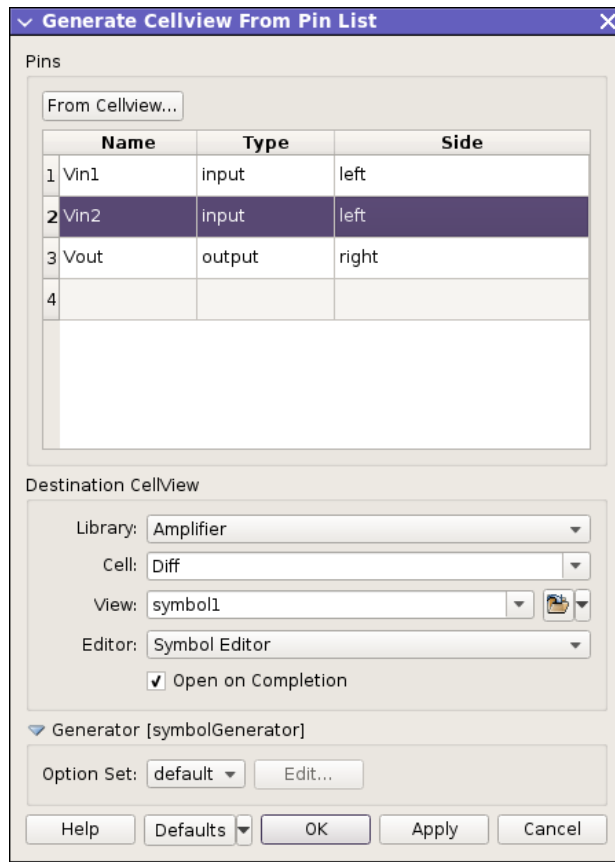


**Question 12.** What is the significance of the red rectangle surrounding the whole symbol?

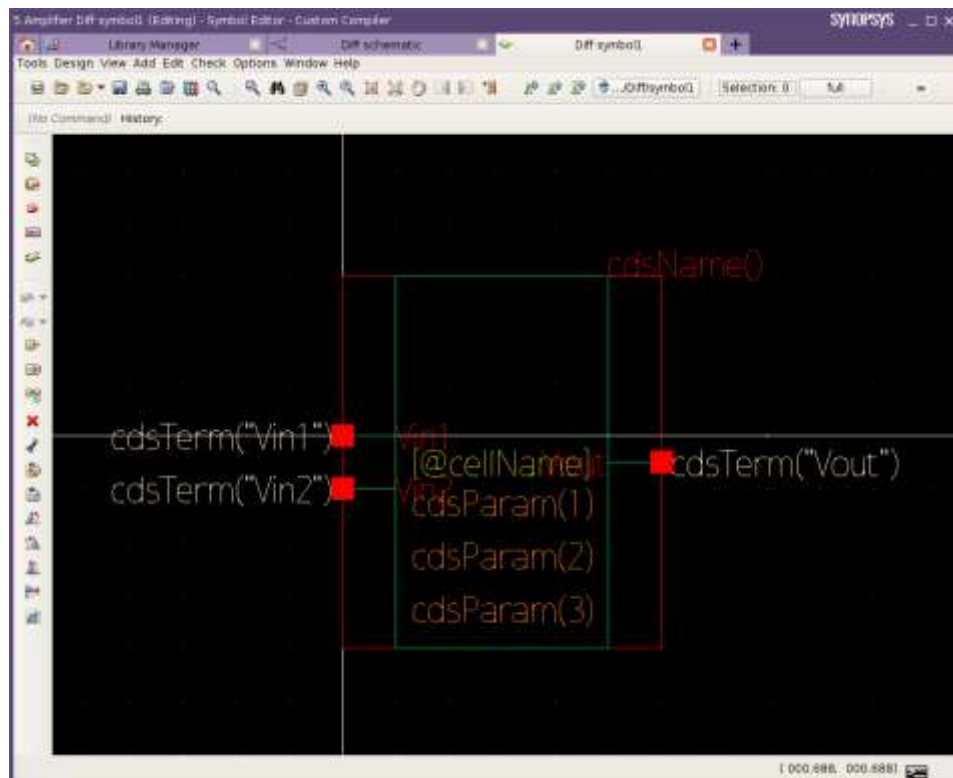
**Note:** The default shape of the symbol is a rectangle with the pins as defined in Step1. The small red squares indicate the connection points for each corresponding pin.

98. Close the Symbol Editor window. From the *Diff* schematic window, create another symbol using **Design → New CellView → From Pins ... (Shift+Y)**. This opens "**Generate Cellview From Pins**" dialog. Fill the dialog fields as depicted in the picture below, specify the destination cellView name as *symbol1*. Click OK button to generate the new symbol.

**Hint:** Read the *Diff* schematic cellView to fill the pin information.



99. The symbol created from **Step 3** will look like as this.



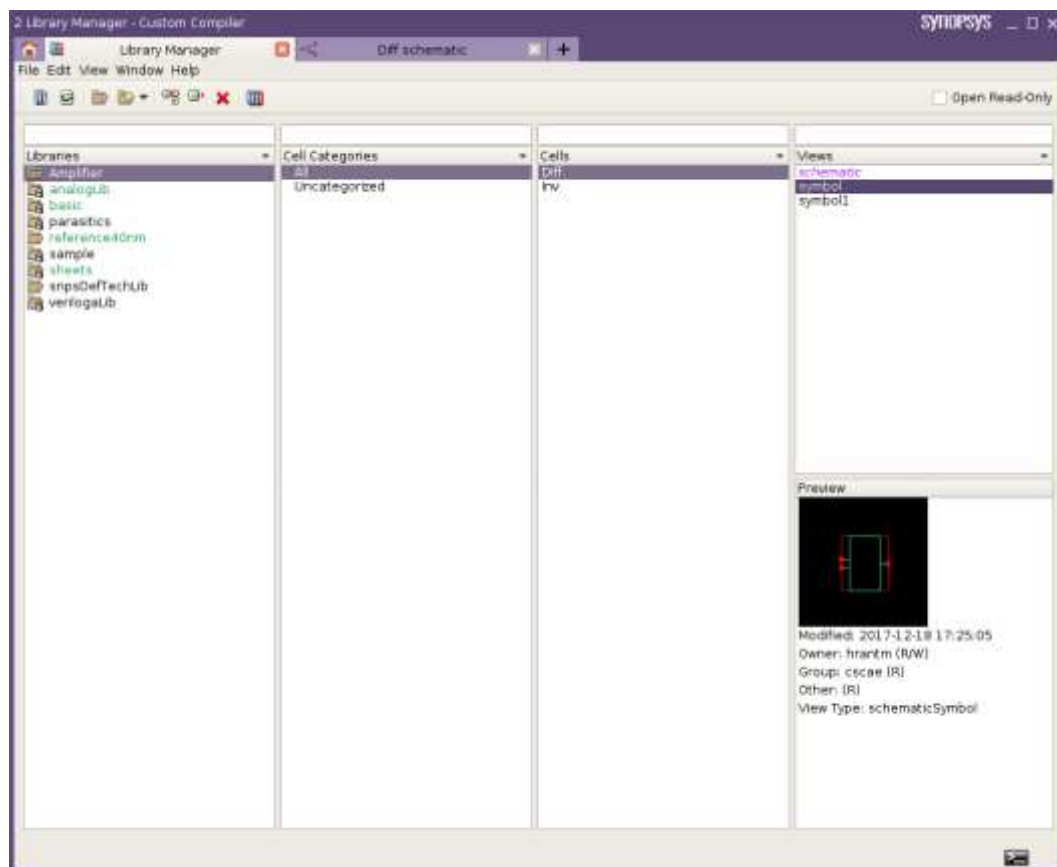
100. Close the *symbol1* window.

## Task 42. Edit the Symbol

Another requirement for the symbols is shapes. It is always a good practice during schematic capture to create symbols whose shapes are more intuitive and reflect their purpose in the schematic.

In this task, you will learn to edit the symbol shape of any one of the symbols generated in Task 4.

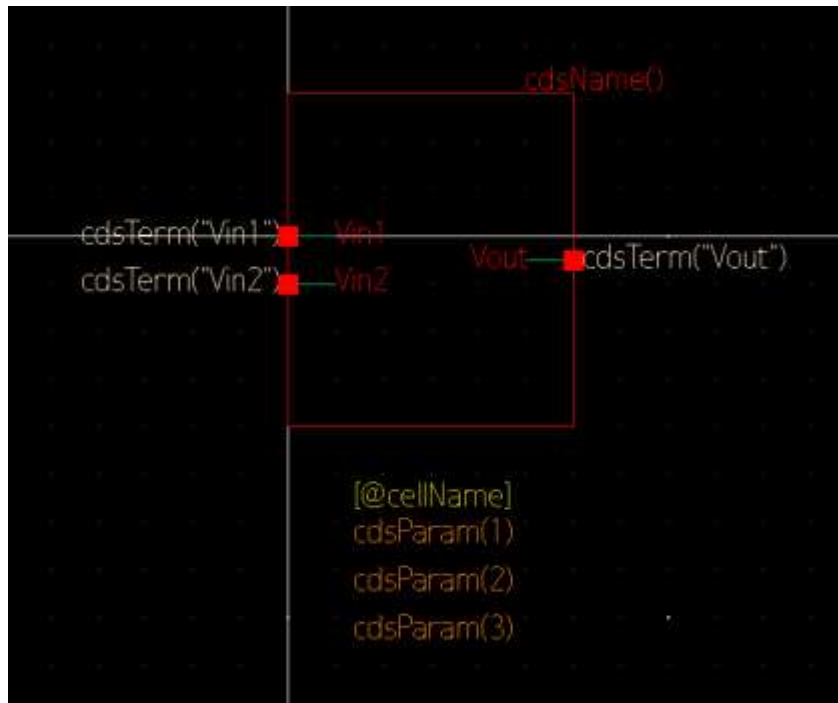
101. Use the symbol which is generated from **Design → New CellView → From CellView ...** for the editing. From Library Manager Window select and open *Diff/symbol* view



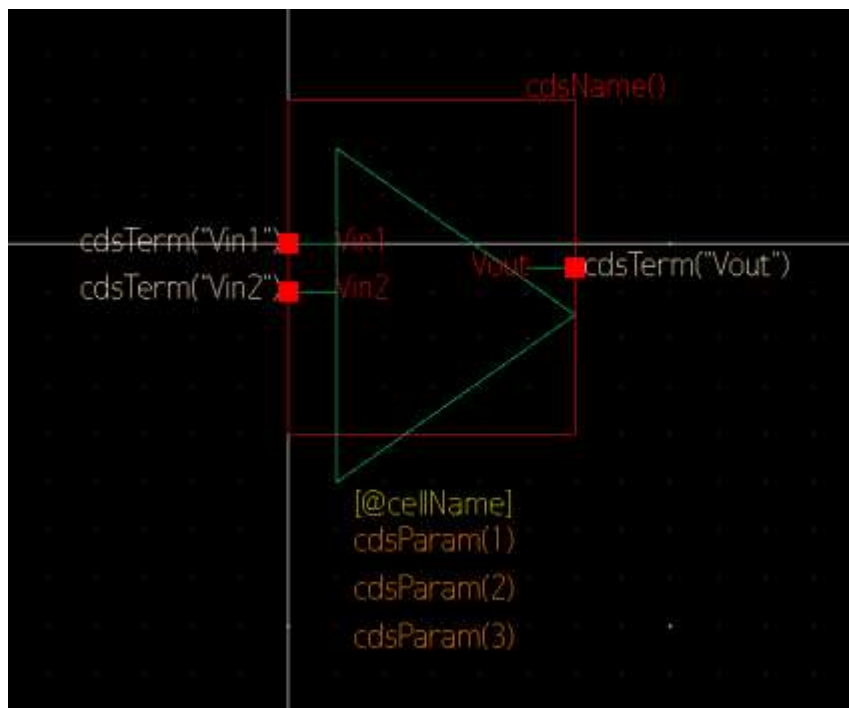
102. Select and delete the green rectangular symbol shape using **Edit → Delete**

103. Move [*@cellName*] and *cdsParam* labels down, until they will not overlap pin labels

**Hint:** Use Shift-click to select multiple objects and **Edit → Move** to move objects.



104. Draw an outline of Differential Amplifier symbol as shown below using **Add → Shape** (Hint: Set the shape icon to “path”)

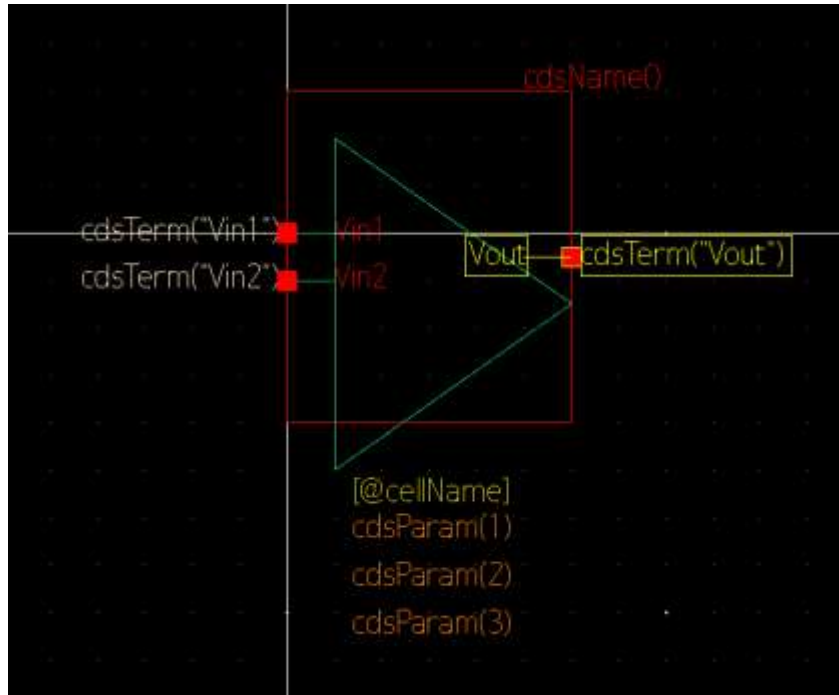


**Note:** During drawing you can delete the last inserted point by pressing “Backspace” button.

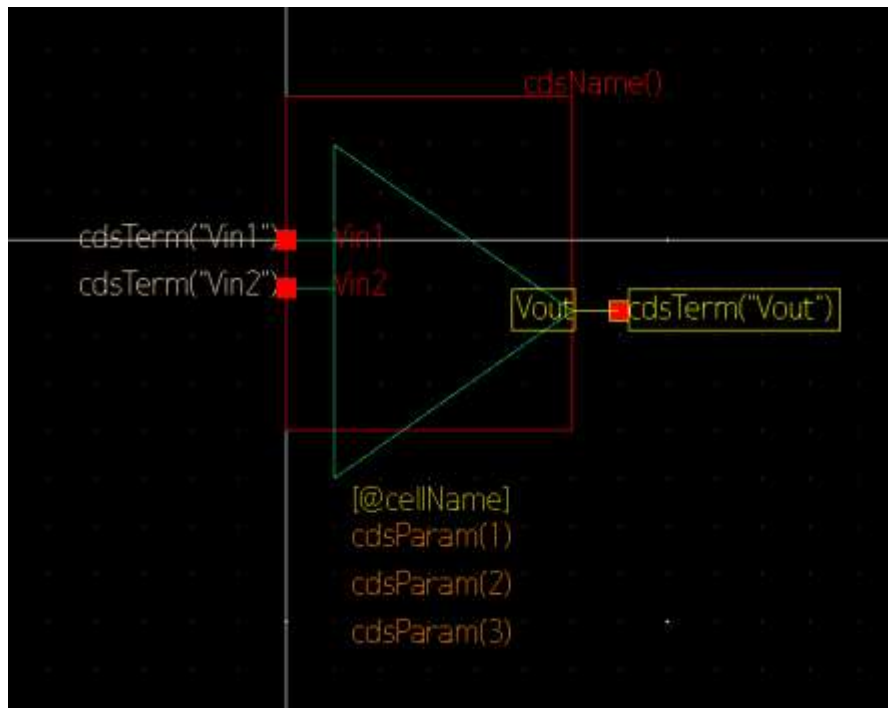
105. Adjust the position of the pins *Vin2* and *Vout* as shown here.

- Select the *Vout* pin with its corresponding labels as shown below

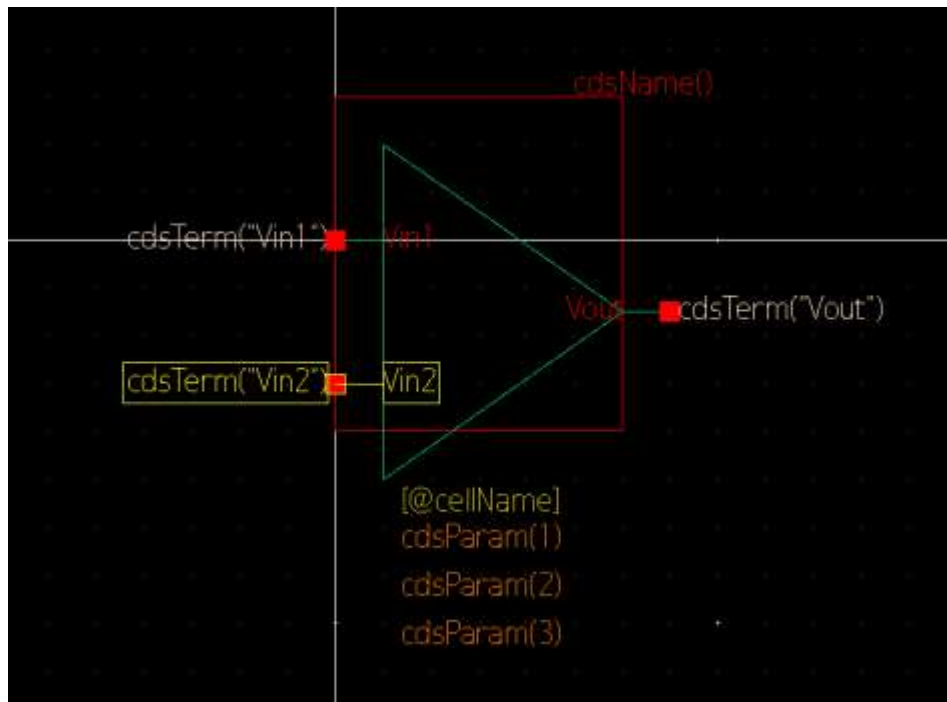
**Hint:** Use Shift-click to select multiple objects.



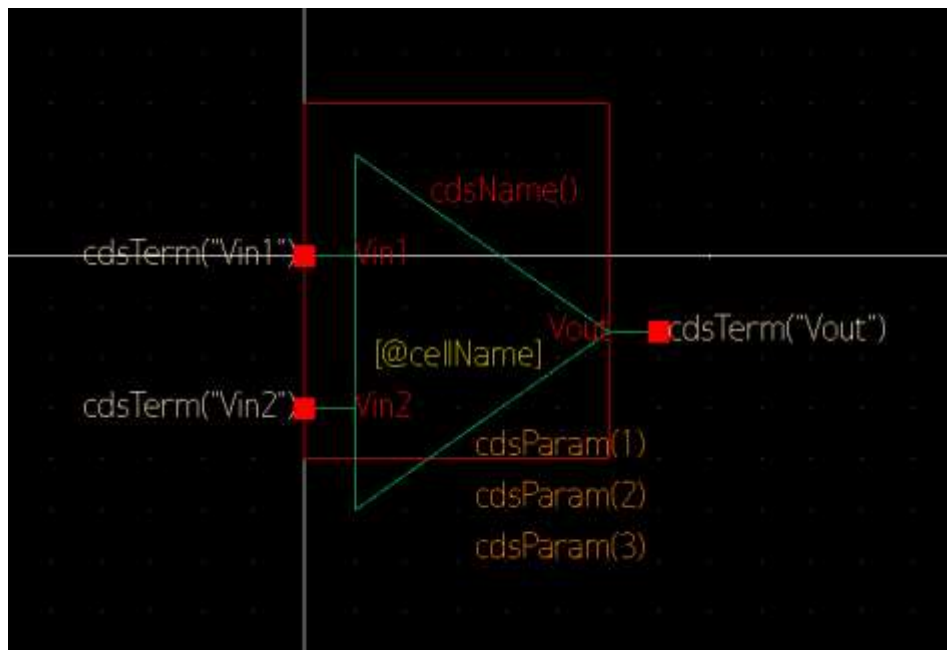
- Place the mouse cursor over the *Vout* pin such that it becomes active.
- Click and drag, and release the mouse button such that your symbol should look like this.



- Similarly adjust the pin *Vin2* as shown:




- 106.** Adjust the positions of other labels using Edit → **Move** to complete the symbol as shown:



## Task 43. Add Selection Shape

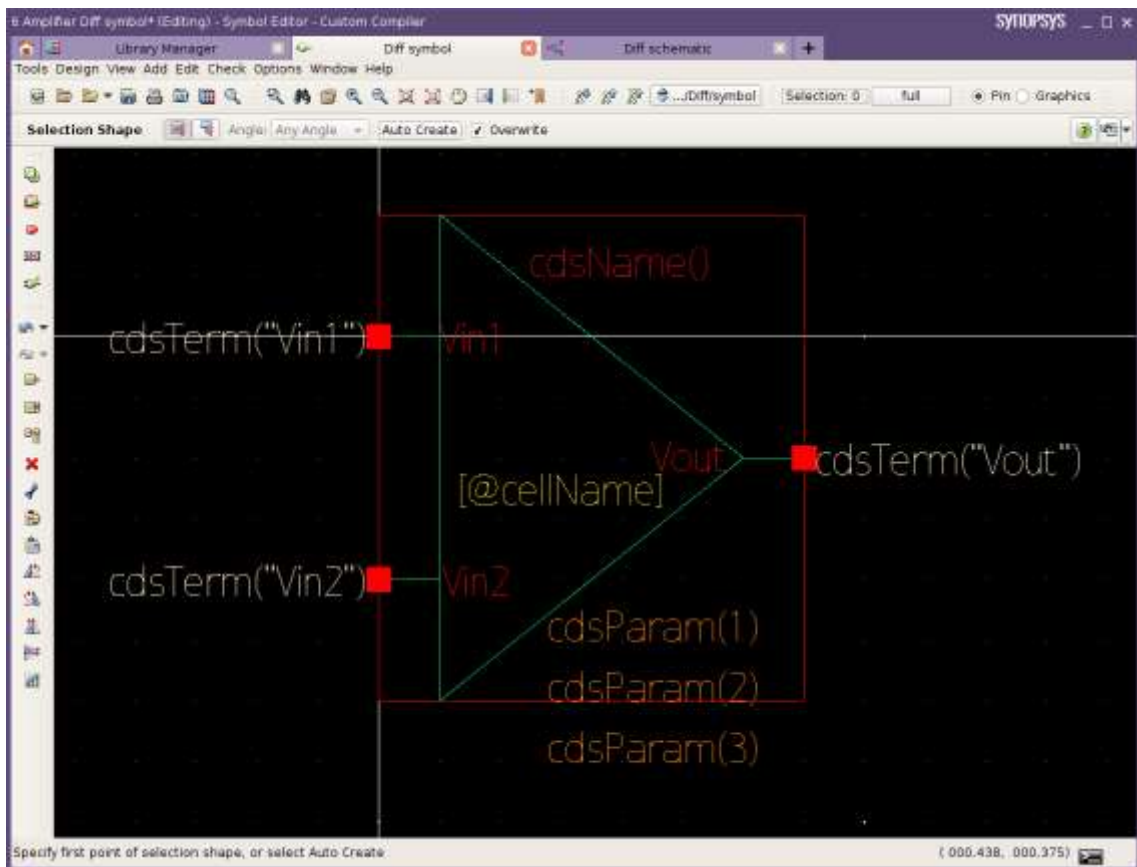
The selection region of the symbol plays an important role when symbols are placed in the schematic. It is defined as an active region which allows you to select the symbol and also defines the routing avoidance area for schematic wiring.

In this task, you will learn to define the selection region for the symbol.

**107.** Select and delete the selection region rectangle using **Edit → Delete** or by clicking on Delete  button on the toolbar.

**108.** Define the new selection region using **Add → Selection (T)** as shown in the image.

**Note:** There are two ways of creating Selection Shape, either drawing the shape manually or by pressing “Auto Create” button.



**Question 13.** On what basis does the **Auto Create** option create the selection shape in the symbol?

**109.** This completes the symbol of CMOS Differential Amplifier Circuit. Save the symbol using **Design → Save**.

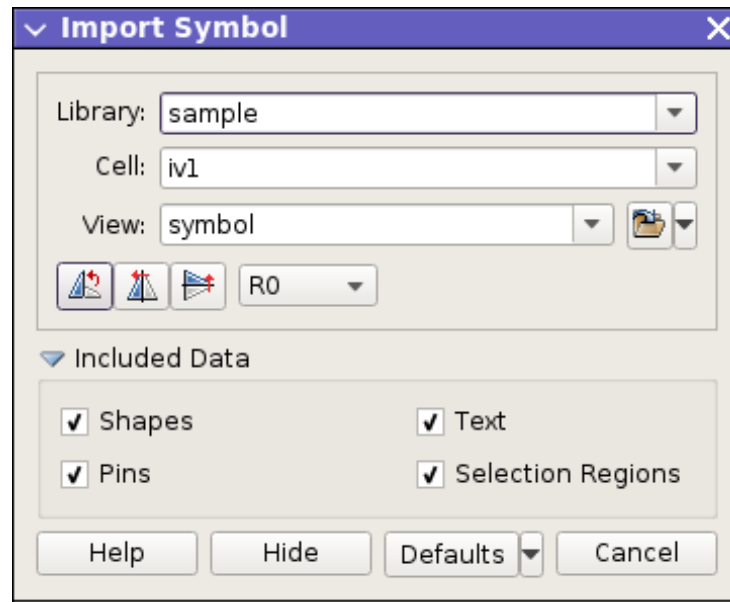
## Task 44. Useful Concept

---

The symbols can be imported from other libraries; this process will save time on creating the symbols.

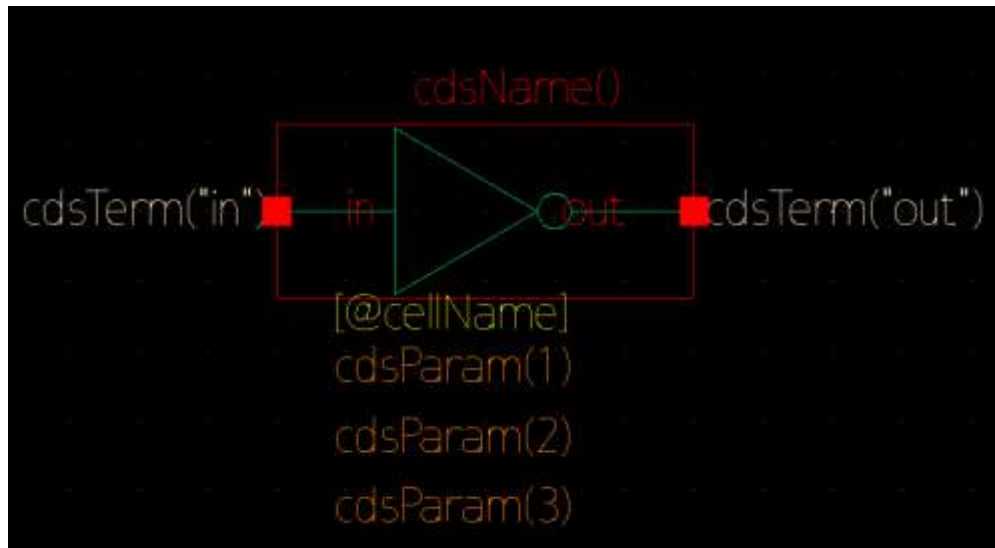
In this task, you will create a symbol view of inverter by importing that symbol from Custom Compiler “sample” symbols library. It contains symbol views of standard cells.

110. From the library Amplifier, open a schematic cell Inv of the view name schematic.
111. Create a symbol cellView with the default settings using **Design→ New CellView → From CellView ... (Y)**.
112. In the opened “Symbol Editor” choose Add → **Import Symbol... (Shift+T)**, and in the opened “Import Symbol” dialog fill the LCV with the values as shown in the picture:

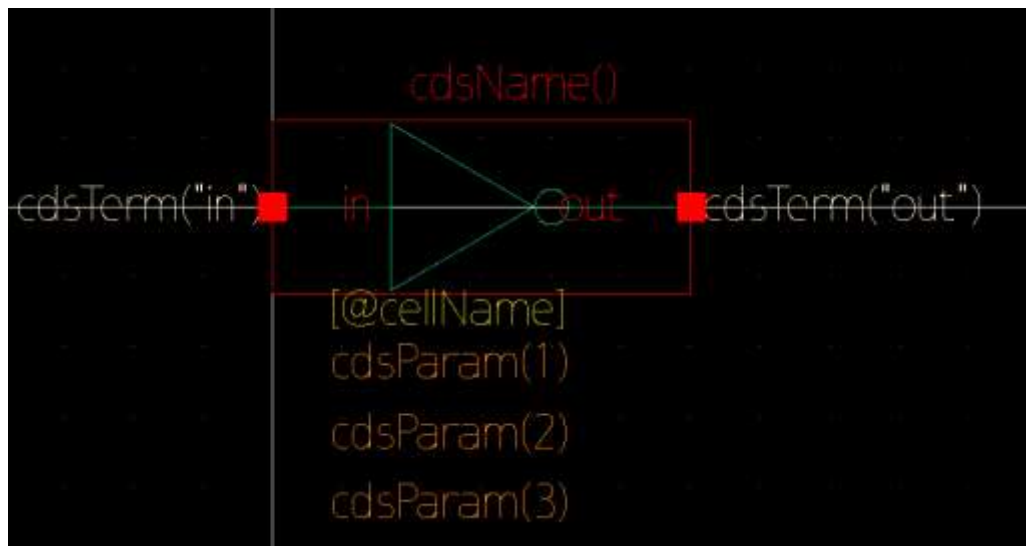


113. Uncheck the “Pins” and “Text” ticks in “Included Data” section of that dialog, for not importing this info with the symbol. The reason of not importing the pins and texts is that their names are different than the pin names in our design.
114. Drag and drop all missing text and pins from CD generated default symbol view to the imported iv1 symbol, and delete the Custom Compiler default generated symbol. The symbol should look like as shown in the picture:





115. Define the symbol origin, the origin setting defines where {0, 0} coordinate is located on the canvas. Choose **Edit → Define Origin**, and click on the left lower corner of the symbol.



116. This completes the symbol of Invertor Circuit. Save the symbol using **Design → Save**.

## Task 45. Check and Save the Symbol

After creating the symbol, it is important to check the symbol for pin name and direction consistency with the corresponding schematic to minimize simulation debugging. It is also important to save your designs frequently so that your database is up-to-date.

In this task, you will learn to verify your symbol and understand the ERC/SRC error checks performed.

**117.** From the library *Amplifier*, open a cell *Diff* of the view name *symbol*. This is the symbol view which was generated on Task 6.

**118.** Perform the check operation using **Check → Cross View Check...**

**Note:** Observe the Console window to see the errors and warnings messages. Also observe the canvas to see if problem markers are generated.

If you have done all the steps correctly, you should not get any errors.

**119.** Save the symbol design using **Design → Save**.

**120.** Delete the output pin *Vout* from the symbol and again perform the ERC/SRC check using **Check → Cross View Check...**

**Question 14.** Why is Cross View Check performed?

**Question 15.** How many errors and warnings are generated in your check and save operation? What are these errors and warnings?

**121.** Now place the output pin *Vout* as deleted in Step 3 using **Add → Pin** or by using **Edit → Undo** if you have not saved your design after the deletion.

**122.** Save the symbol design.

**123.** Again, perform the cross view check operation and check the Console for any errors and warnings messages.

**Question 16.** What message did you get after fixing the above errors and warnings?

**124.** You have successfully verified your symbol. Save and close all the opened designs.

## ***Congratulations!***

You have successfully created the symbol of the CMOS Differential Amplifier Circuit!

# Answers / Solutions

## Task 12. Create a Symbol

**Question 12.** How many pins are created in the symbol cellView and what are their names?

**125.** There are three pins created with the names *Vin1*, *Vin2* and *Vout*.

**Question 13.** What is the significance of the red rectangle surrounding the whole symbol?

**126.** When the symbols are used in the schematics, this red rectangle surrounding the whole symbol determines the selection area to select the symbol and also defines the routing avoidance area for the schematic wiring.

## Task 6. Add Selection Shape

**Question 14.** On what basis does the **Auto Create** option create the selection shape in the symbol?

**127.** Symbol pins are the basis for the **Auto Create** option to define the selection shape.

**128.** The selection shape will be a minimum sized rectangle enclosing the center point of all the symbol pins.

## Task 7. Check the Symbol

**Question 15.** Why is Cross View Check performed?

**129.** Cross view check is generally performed for ensuring pin name and direction consistency between different views. (In this case it is schematic and symbol)

**Question 16.** How many errors and warnings are generated in your check and save operation? What are these errors and warnings?

**130.** Ideally, there should be 1 Error, if you have done each step successfully.

**131.** The following error must be returned on the Console:

*Cross view checking for Amplifier/Diff/schematic with Amplifier/Diff/symbol*

*Extra Terminal*

*Extra terminal Vout found in schematic not found in symbol*

**Question 17.** What message did you get after fixing the above errors and warnings?

**132.** If there is no violation in your symbol, you will get the following message in your Console window.

*Cross view checking for Amplifier/Diff/schematic with Amplifier/Diff/symbol*

*No rule violations found*

# 6

## 4. Hierarchical Data Creation and Design Navigation

### Learning Objectives

During this lab, you will learn to build and traverse through design hierarchy using the various features of the Custom Compiler Schematic Editor.

After completing this lab, you should be able to:

- Create a design hierarchy using symbols.
- Traverse the hierarchy.
- Use Bookmarks and Edit in Place features to quickly access and edit design information.

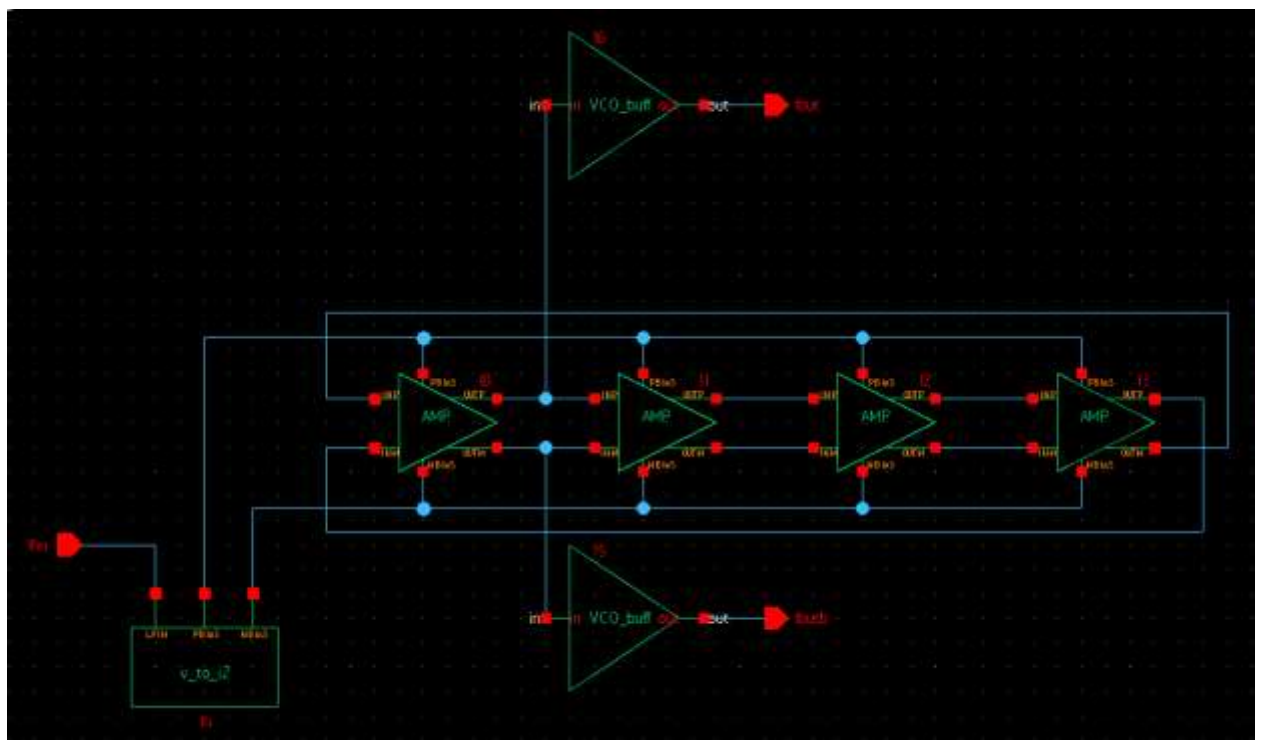


**Lab Duration:**  
45 minutes

# Introduction

Design hierarchy consists of many levels of a single design. Each time you descend into the design hierarchy, you view a smaller module of the design in greater detail. Working in smaller modules of a design lets you distribute work across multiple design engineers or groups more effectively.

In this lab, you will create a hierarchical schematic of a *VCO* (Voltage Controlled Oscillator) design as shown below by using symbols of lower level schematics. First you will create the buffer design *VCO\_buff* (using basic building blocks) which will be added to the *VCO* design.

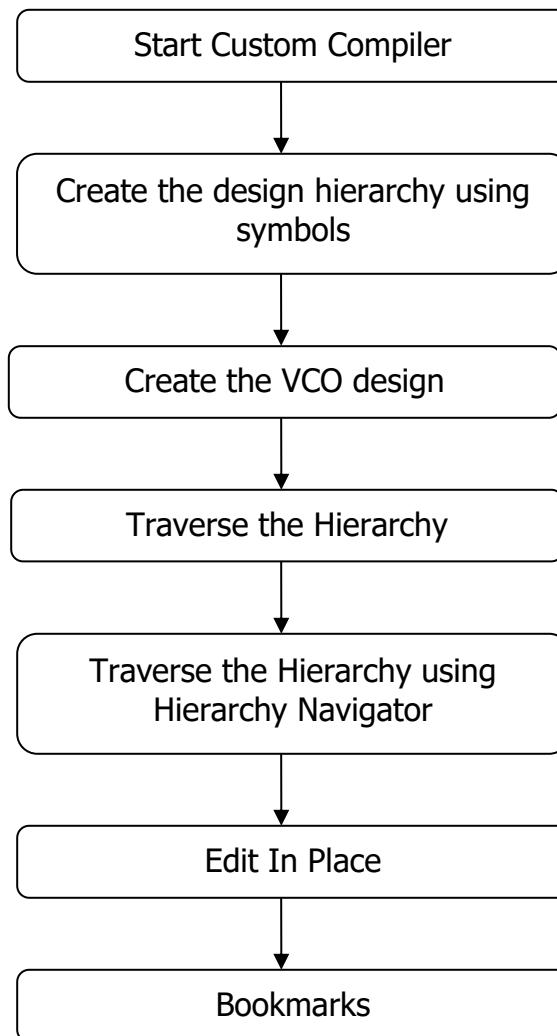


In this lab, you are provided with the *Design* Open Access database library and a *lib.defs* file. The *Design* library contains the schematic and the symbol of the basic building blocks to design the *VCO* schematic. You will use these symbols to create your hierarchical schematic.

Please see the section “Data Creation”, “Editing”, “Symbol Generation”, “Hierarchy Traversal” and “Bookmarks” of the *SE command reference manual* for the commands that are used in this lab.

# Flow Overview

## Lab1 Tasks



# File Locations

All files for this lab are located in the directory *SE\_Design\_Hierarchy\_Lab1*.

## Directory Structure

|                                       |                           |
|---------------------------------------|---------------------------|
| <code>SE_Design_Hierarchy_Lab1</code> | Current working directory |
| <code>Design</code>                   | OpenAccess Design library |
| <code>lib.defs</code>                 | Library Definitions file  |

## Relevant Files

|                     |                 |
|---------------------|-----------------|
| <code>../PDK</code> | Reference40 PDK |
|---------------------|-----------------|

## Answers & Solutions

There is an *ANSWERS / SOLUTIONS* section at the end of this lab. You are **encouraged** to refer to this section often to verify your answers, or to obtain help with the execution of some steps.



# Instructions

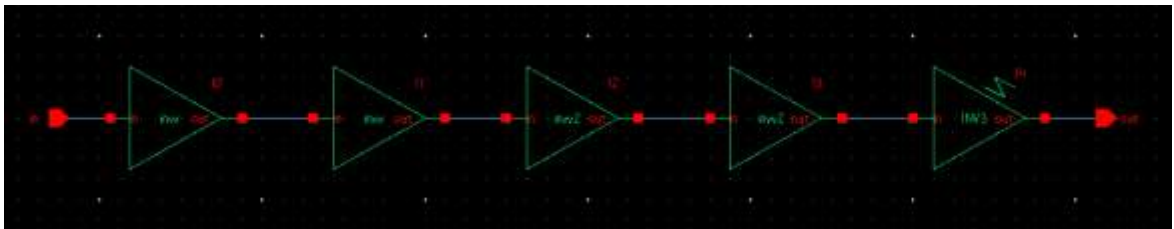
## Task 46. Start Custom Compiler

133. In the Unix terminal window change your current working directory to *SE\_Design\_Hierarchy\_Lab1*. This will be your working directory for this lab.
134. Start Custom Compiler from the Unix prompt.

```
custom_compiler &
```

## Task 47. Creating the design hierarchy using symbols

In this task you will create the buffer design *VCO\_buff* as shown below using the basic inverter blocks (*invv*, *invv2*, *INV3*) from the *Design* library.



**Note:** The inverters *invv*, *invv2* and *INV3* used here have different current driving capabilities.

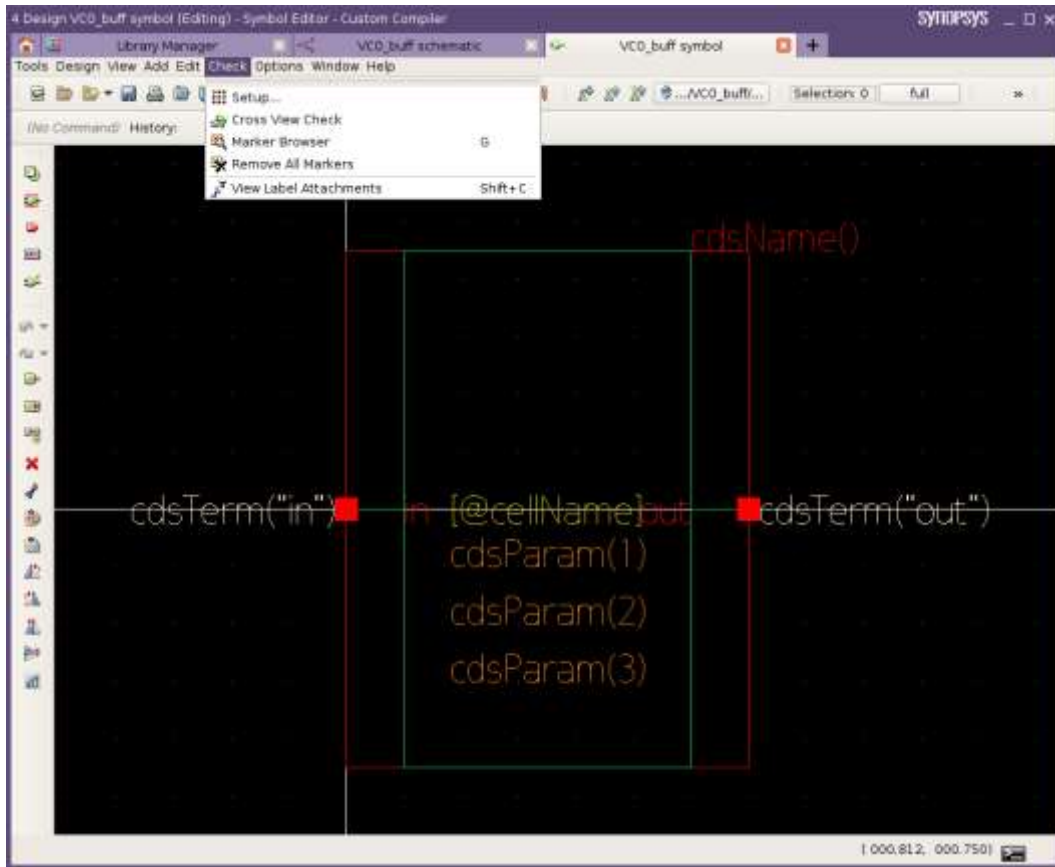
135. In the library *Design*, create a schematic cell *VCO\_buff* of the view name *schematic*.

**Note:** The *VCO\_buff* cellView will be opened in a Schematic Editor window.

136. Place the basic inverter symbols *invv* (*I0*, *I1*), *invv2* (*I2*), *INV3* (*I3*) from the library *Design* using **Add > Instance...** as shown.
137. Using the image as a reference, place an input pin *in* and output pin *out* using **Add > Pin**.
138. Complete the wiring of the instances using **Add > Wire** as shown.
139. Check and save the schematic using **Design > Check and Save**. Check the Console for any errors or warnings  
  
This completes the design of *VCO\_buff*. The next step is to create the symbol of *VCO\_buff* which will be used to create the *VCO* design.
140. Create a symbol cellView with the default settings using **Design > New CellView > From CellView ...** such that the input pins are on the left side and the output pins are on the right side.

**Hint:** Use the **Adjust Pins option** to modify the pin positions.

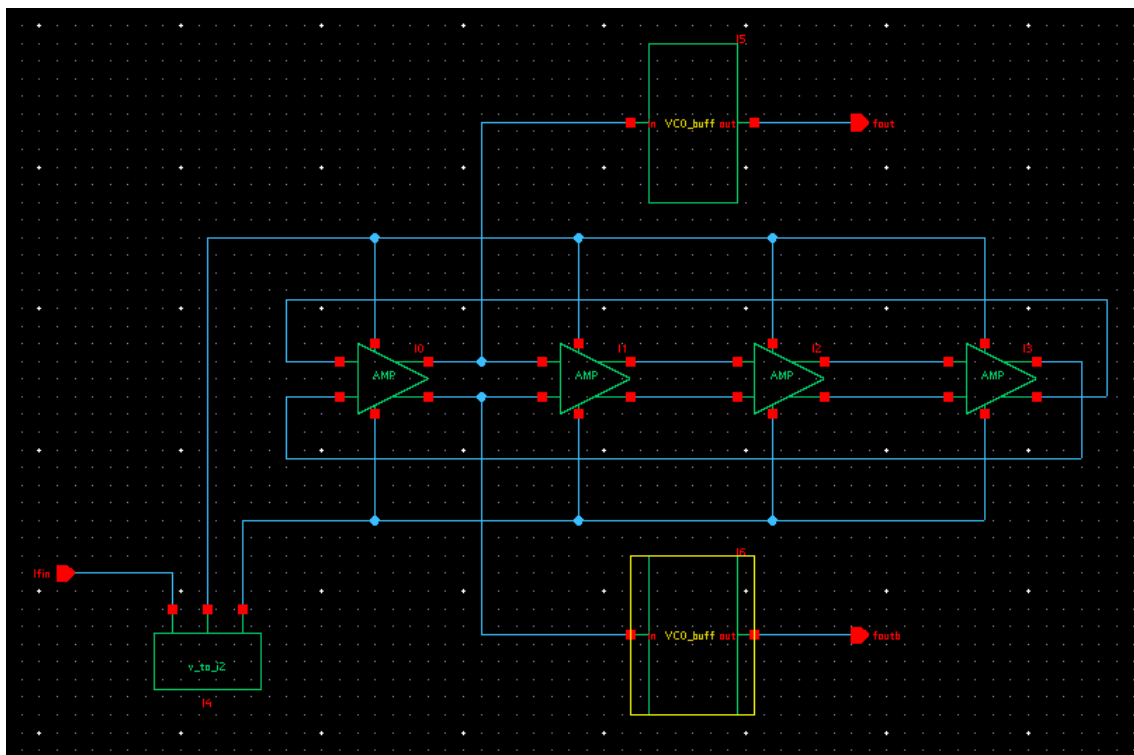
141. Check the symbol using **Check > Cross View Check ....** Check the Console for any errors or warnings.



142. Save the symbol.
143. Close all the open designs.

## Task 48. Creating VCO Design

In this task you will create the next level of the hierarchy by creating the VCO design as shown below using the design blocks amplifier *AMP*, voltage to current converter *v\_to\_i2*, buffer *VCO\_buff*.



144. In the library *Design*, create a schematic cell *VCO* with the view name *schematic*.
145. Place the design symbols *AMP* (*I0*, *I1*, *I2*, *I3*), *v\_to\_i2* (*I4*), *VCO\_buff* (*I5*, *I6*) from the library *Design* using **Add > Instance...** as shown.
146. Using the image as a reference, place an input pin *lfin* and output pins *fout*, *fouth* using **Add > Pin**.
147. Complete the wiring of the instances using **Add > Wire** as shown in the image.
148. Check and save the schematic using **Design > Check and Save**. Check the Console for any errors or warnings.

This completes the *VCO* design with two levels of hierarchy.

## Task 49. Traversing the Hierarchy

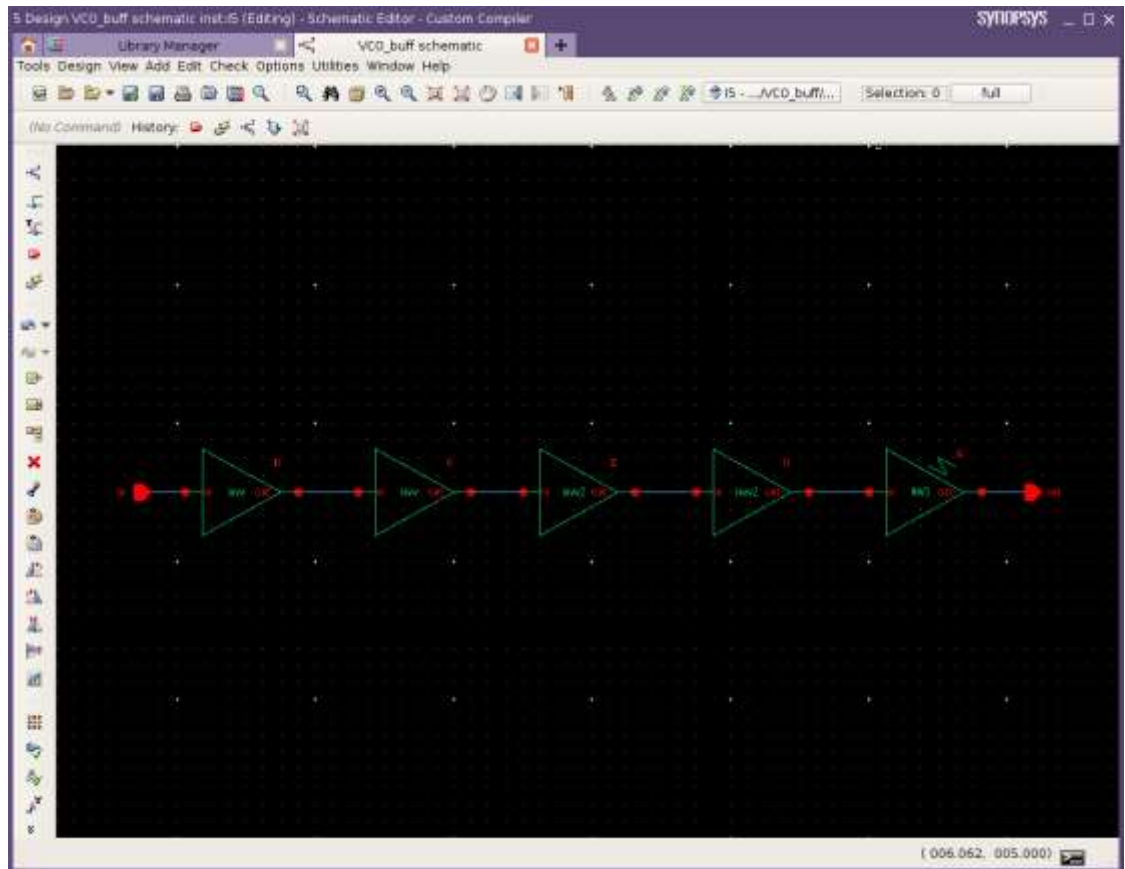
When you have a complex hierarchical schematic you may want to make modifications in different cells without having to close and open separate windows. You can do that by traversing the hierarchy. You can traverse both up and down the hierarchy, make modifications, and then go back again.

In this task you will traverse down the hierarchy to do necessary modifications in the design and then return to the top of the design.

149. Select the *VCO\_buff* instance *I5* in the *VCO* design and descend one level down the hierarchy using **Design > Hierarchy > Descend Edit**.

**Note:** The schematic cellView *VCO\_buff* is opened in the same window in the edit mode. (Now you are one level down the hierarchy)

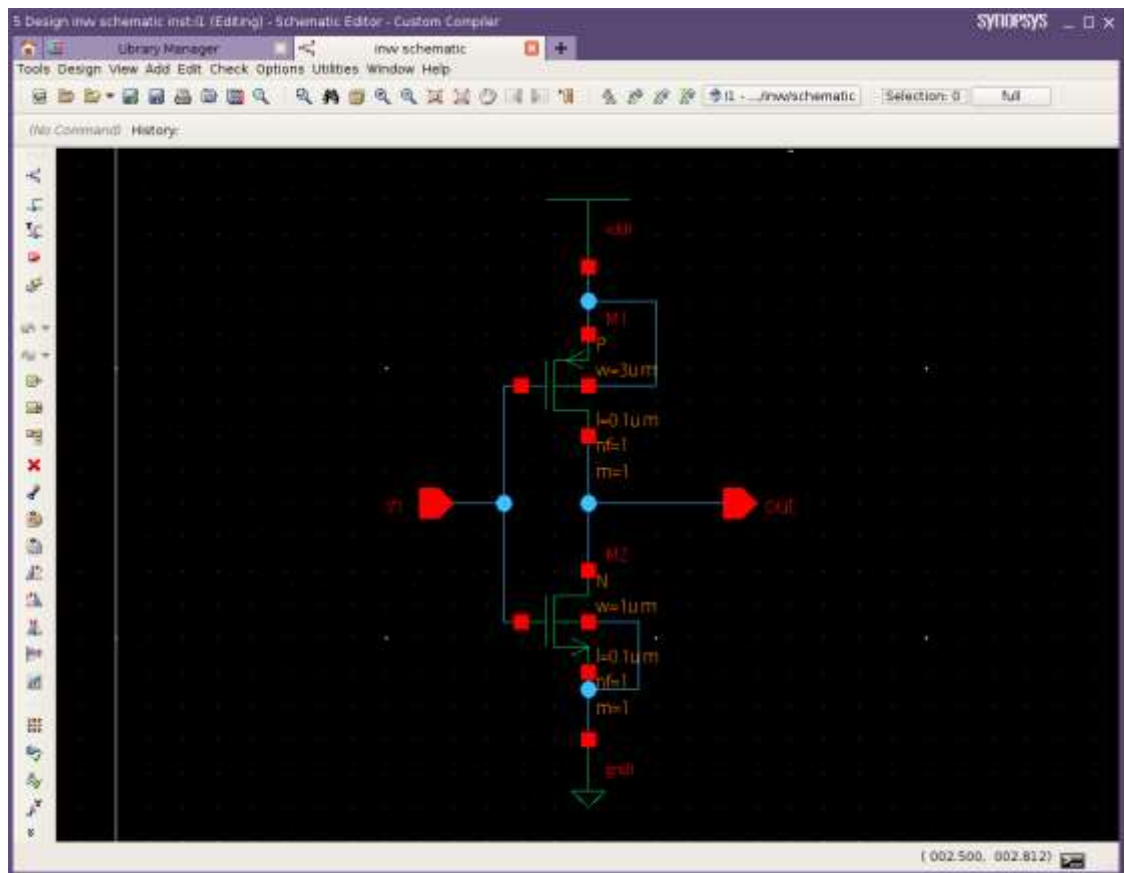
**Question 18.** What is the other method which quickly takes you one level down the hierarchy without accessing command from the menu?



**150.** Select the *invv* instance *I1* in the *VCO\_buff* design and descend one more level down the hierarchy.

**Note:** The schematic cellView *invv* is opened in the same window in the edit mode. (Now you are two levels down in hierarchy)

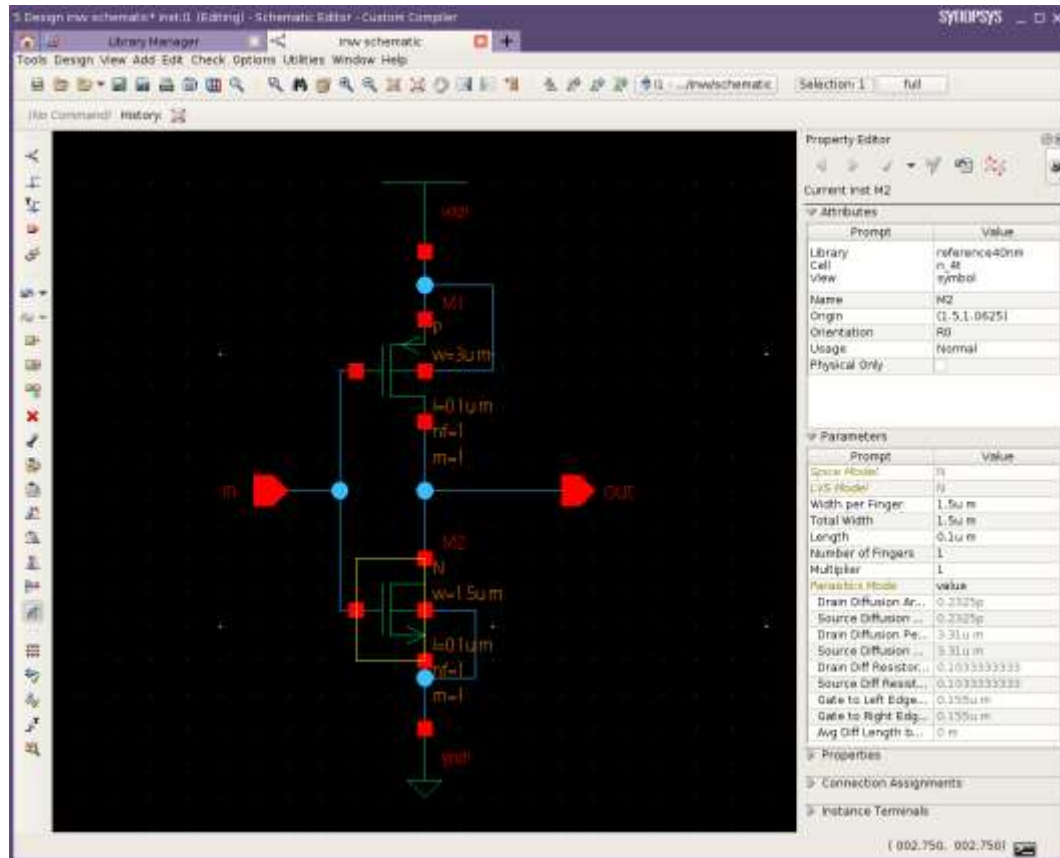
**Question 19.** How you can descend down the hierarchy in the read mode?



151. Modify the parameter value of the nmos instance *M2* using **Edit > Properties > Property Editor**.

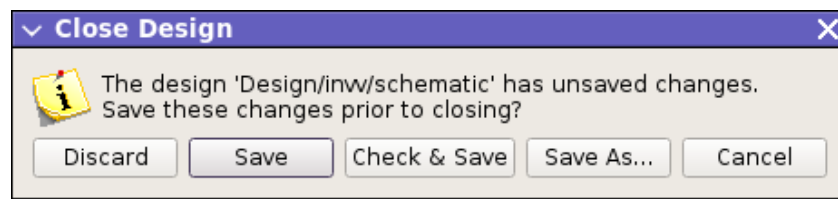
a) Width value to 1.5u

**Note:** This property change will apply to all *invv* instances in your hierarchical schematic.



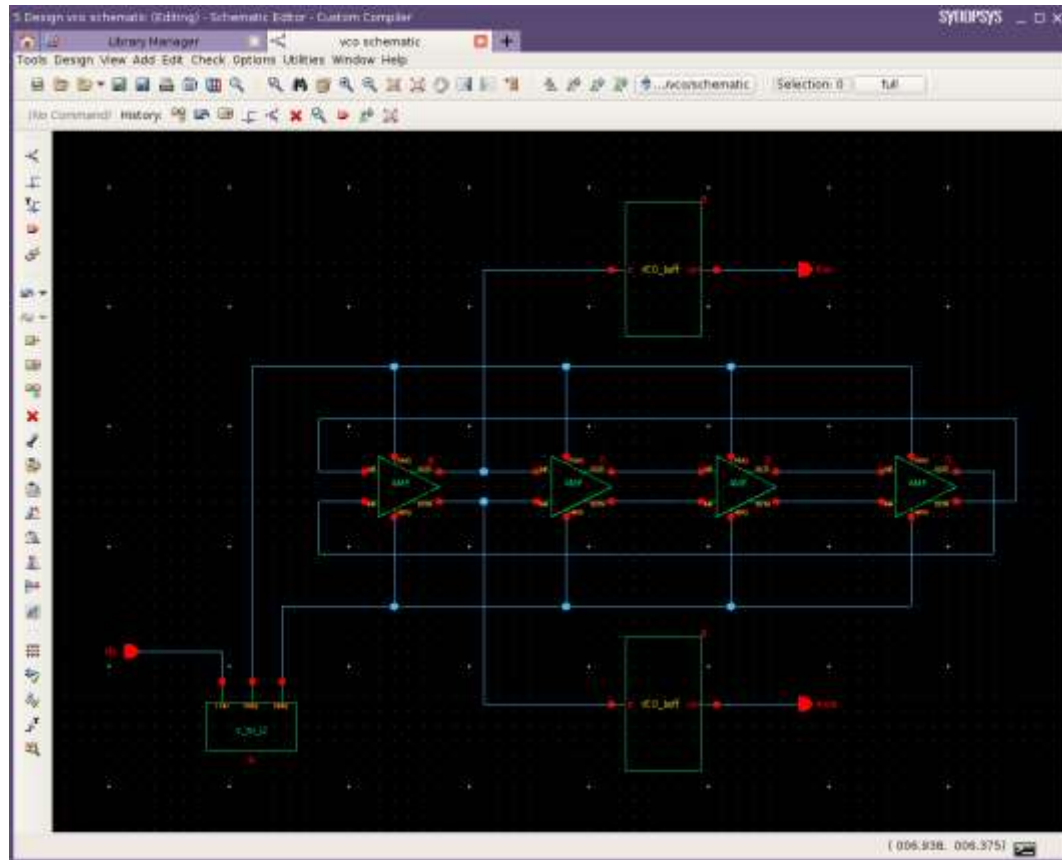
152. Try to return to the top level of the hierarchy using **Design > Hierarchy > Return to Top** or by using the bind key **Shift-b**.

**Note:** You will be prompted to save the design before returning to the top.



153. Click **Save**.

**Note:** You will return back to top of the hierarchy, that is, *VCO* schematic cellView.

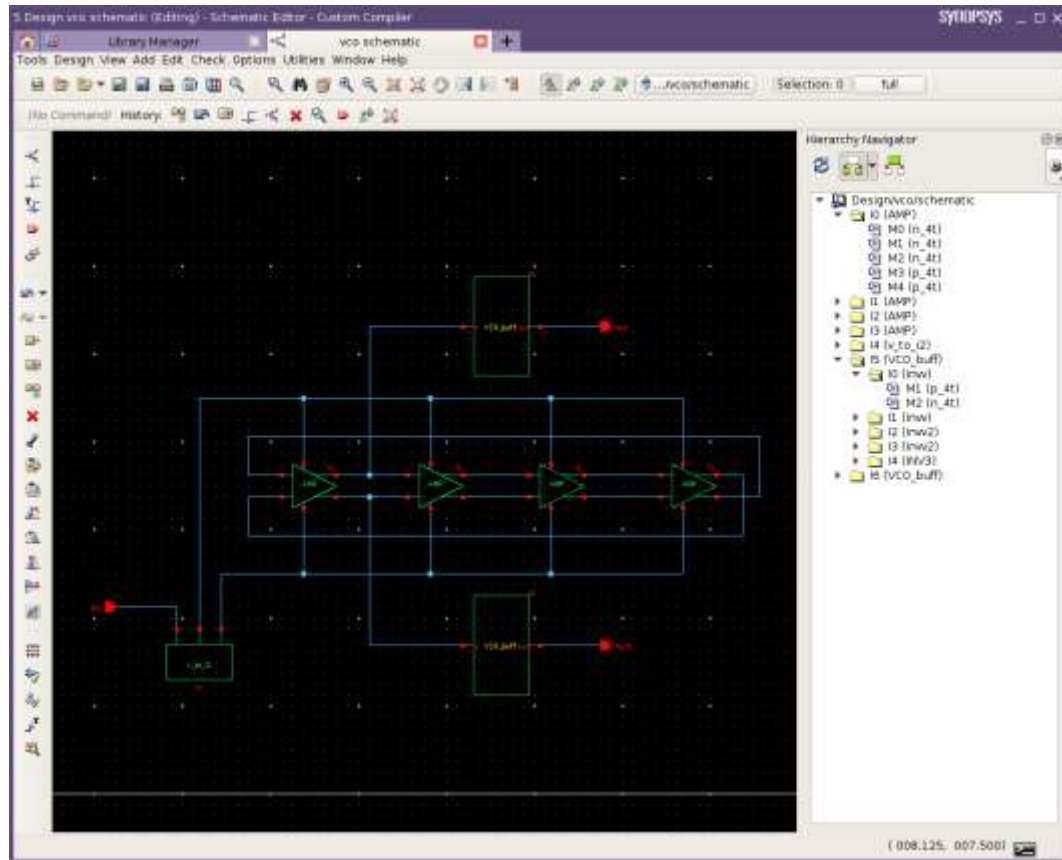


## Task 50. Navigating the Hierarchy using Hierarchy Navigator

The Hierarchy Navigator is a dockable assistant that gives you a tree view of the design hierarchy. You will be able to navigate through the design hierarchy by double clicking on entries in the Hierarchy Navigator. The Hierarchy Navigator is automatically updated whenever the design hierarchy changes.

154. Invoke the Hierarchy Navigator from the VCO schematic cellView window using **Window > Assistants > Hierarchy Navigator** or by pressing the bind key **Shift-T**.

**Note:** The Hierarchy Navigator is docked to the right side of the schematic window. By default, it shows the Hierarchy tree by instance.



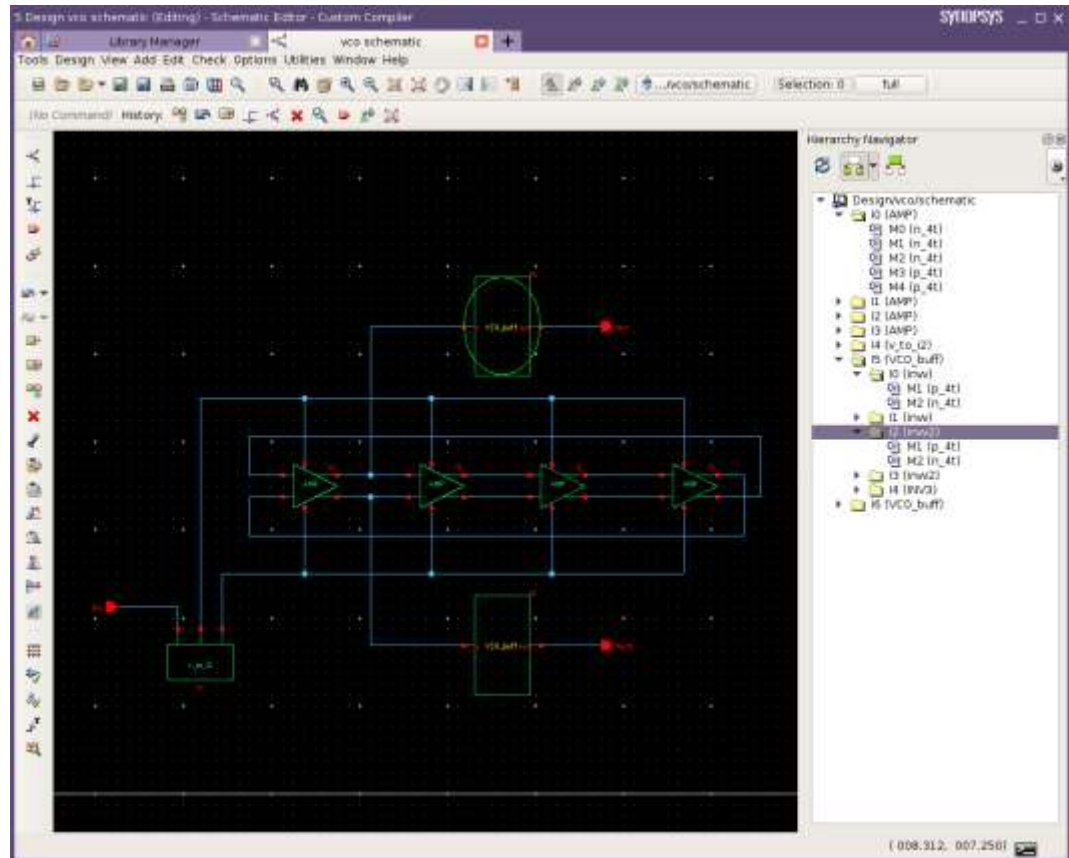
155. Click on the *VCO* lineage to expand the hierarchy.

**Note:** By default, it shows the hierarchy by instance.

**Question 20.** What do you understand from *I5 (Design/VCO\_buff/schematic)*?

156. Similarly, click on + sign in front of *I5(Design/VCO\_buff/schematic)* lineage to expand the hierarchy.



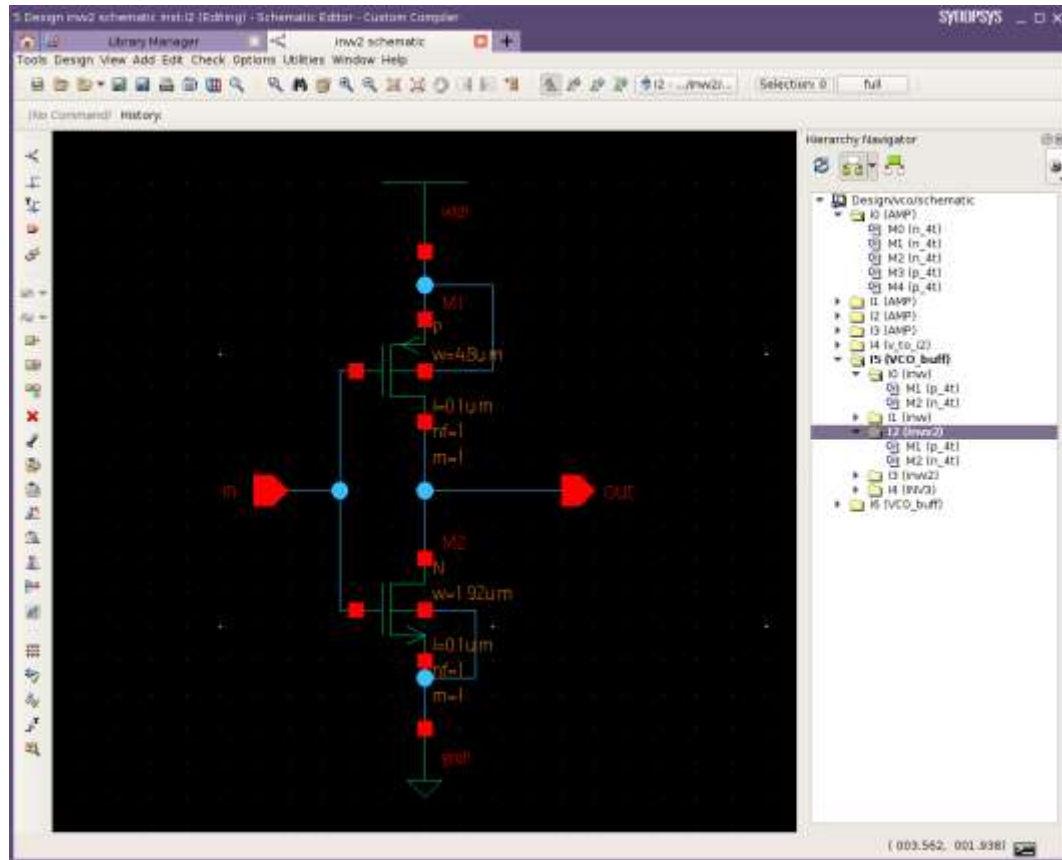


**Note:** I5 instance will be highlighted with green oval.

**157.** Now, Double-click on *I2 (Design/invv2/schematic)* under *I5*.

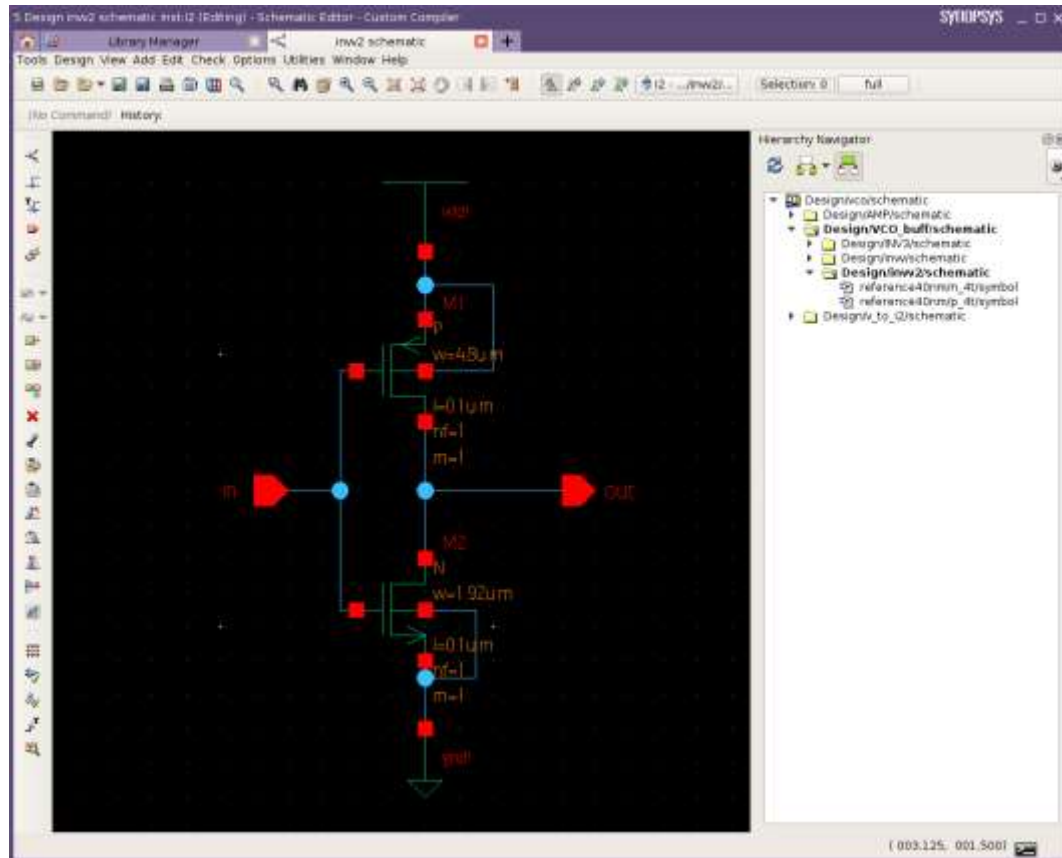
**Note:** This will directly descend into the *invv2* design in a same window. In other words it has directly descended into instance *I2 (Design/invv2/schematic)* from the top which is two levels down the hierarchy.

At the same time, the parent cell *I5 (Design/VCO\_buff/schematic)* and the current cell *I2 (Design/invv2/schematic)* in the hierarchy tree becomes bold to make you understand at what level you are in the hierarchy.



158. Change the Hierarchy Navigator view by clicking on Hierarchy by Master button.



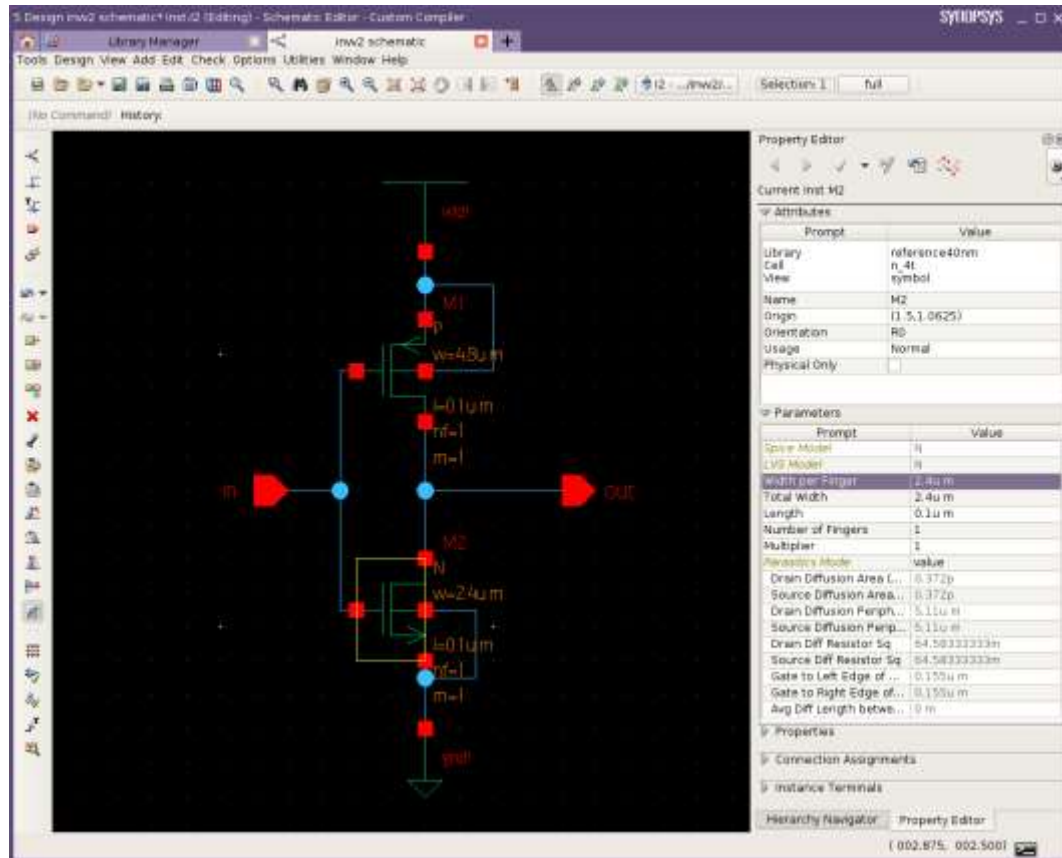


159. Double-click on *Design/inv2/schematic*.

**Note:** The *invv2* cellView opens in a new Schematic Editor window.

160. Modify the parameter value of the nmos instance *M2* in the *invv2* cellView window using **Edit > Properties > Property Editor**.

b) Width value to 2.4u.



161. Save and close the *invv2* design.
162. In the Schematic Editor window, return to the top level using the bind key **Shift-b**.
163. Close the Hierarchy Navigator in the *VCO* design Schematic Editor window.

## Task 51. Modifying symbol representation using Edit in Place

The Edit in Place feature enables you to modify the symbols in the design hierarchy without opening the symbol in a separate window.

In this task, you will modify the rectangular *VCO\_buff* symbol in the *VCO* design.

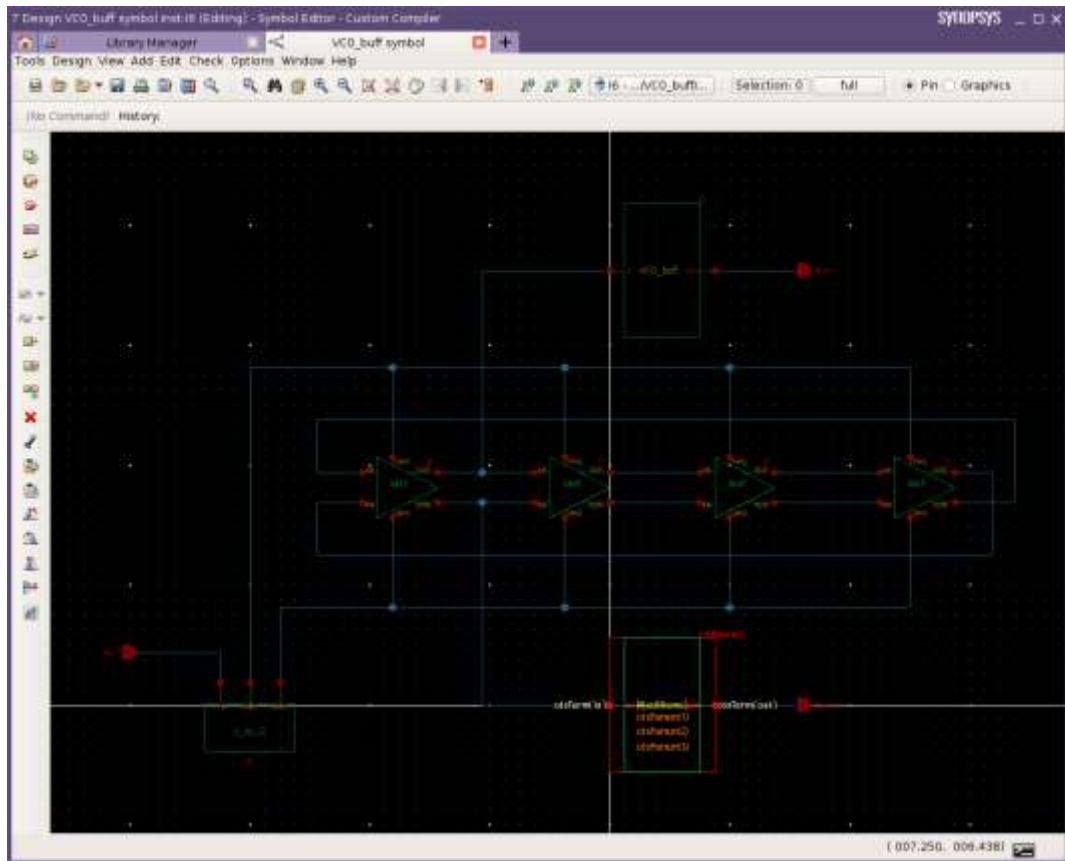
164. Select the *VCO\_buff* instance *I6* and choose **Edit in Place** from **Design > Hierarchy**.

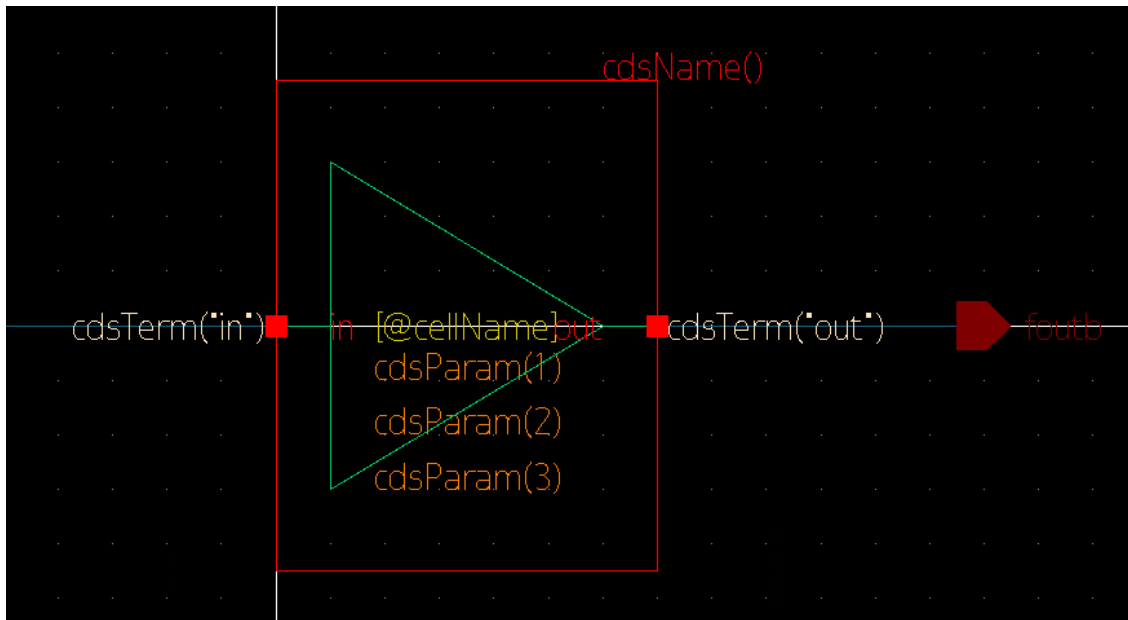
**Note:** This will open up the Symbol Editor in the same schematic window as shown to edit the symbol of this cell.

**Question 21.** Can you find out which are the instances you can edit?

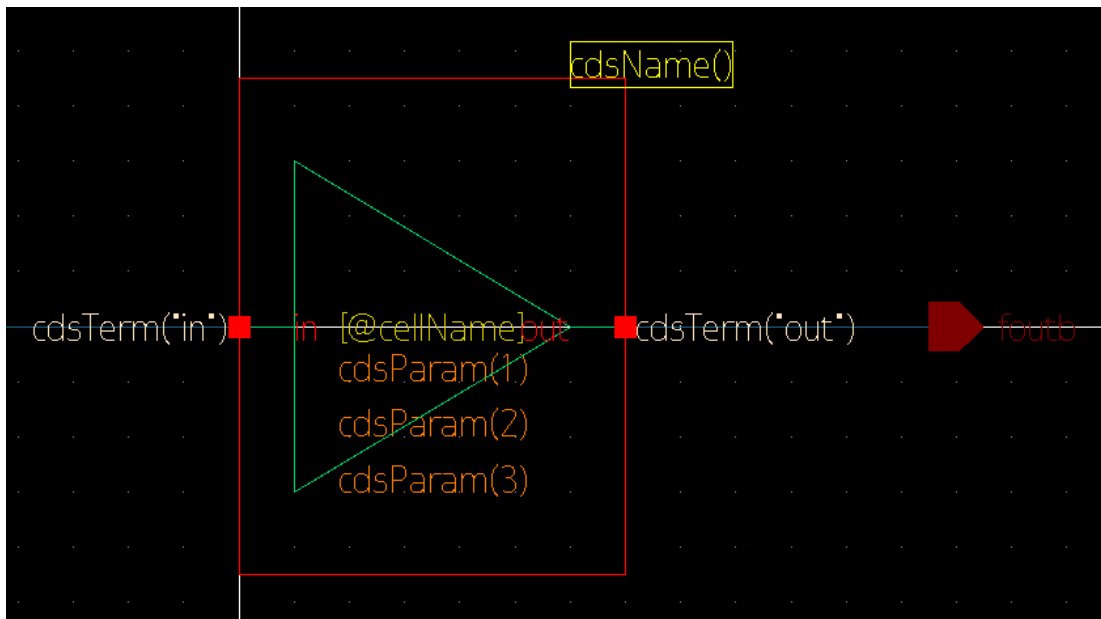
- .....
165. Change the rectangular symbol shape of instance *I6* of *VCO\_buff* into a triangle one as shown in the image below.

**Note:** Observe that these changes apply to all *VCO\_buff* instances in the design. In this case it is instance *I5* of *VCO\_buff*.

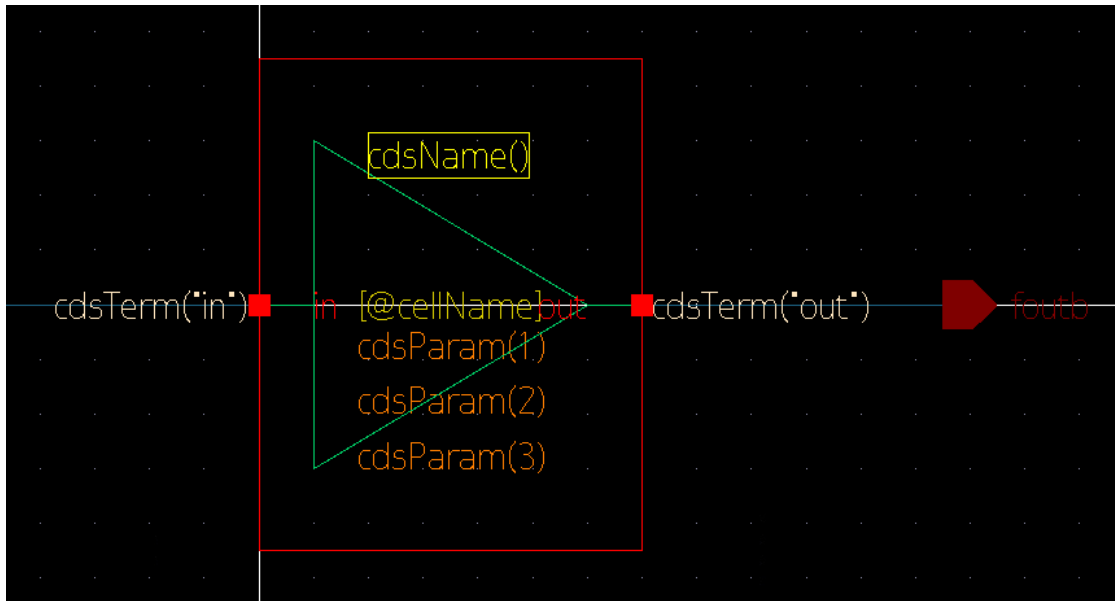




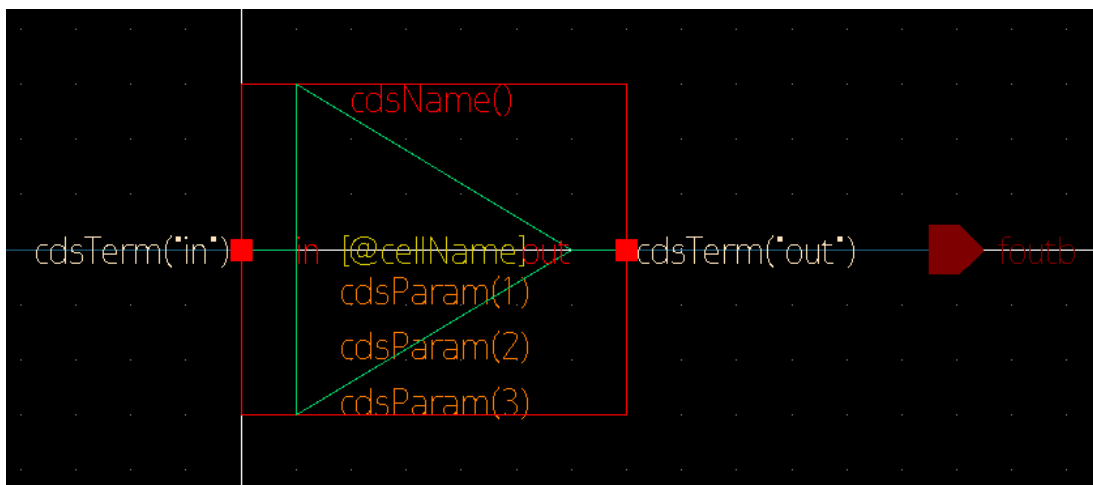
166. Select the label as shown



167. Move the labels using **Edit > Move** to a new position as shown



**168.** Change the selection shape using **Add > Selection**

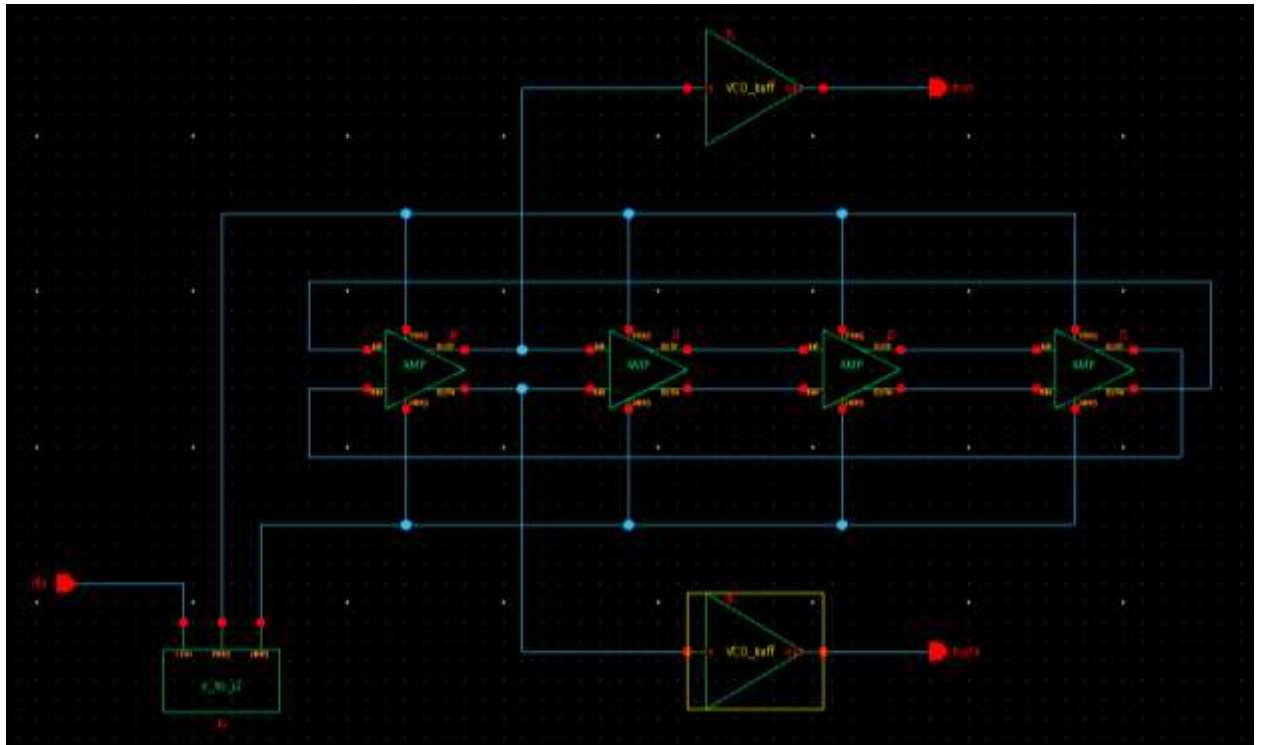


**Note:** The selection shape can be added by pressing “Auto Create” button in the Selection Shape COT.

**169.** Return to the top level of the hierarchy using **Design > Hierarchy > Return To Top.**

**Note:** You will be prompted to save the design before returning to the top. Click **Save**.

**170.** The final schematic looks like this.



## Task 52. Using Bookmarks

In the design process, there are instances where you access the design back and forth to get the information from the design or to modify the information of the design. It is always difficult to frequently traverse up and down the hierarchy.

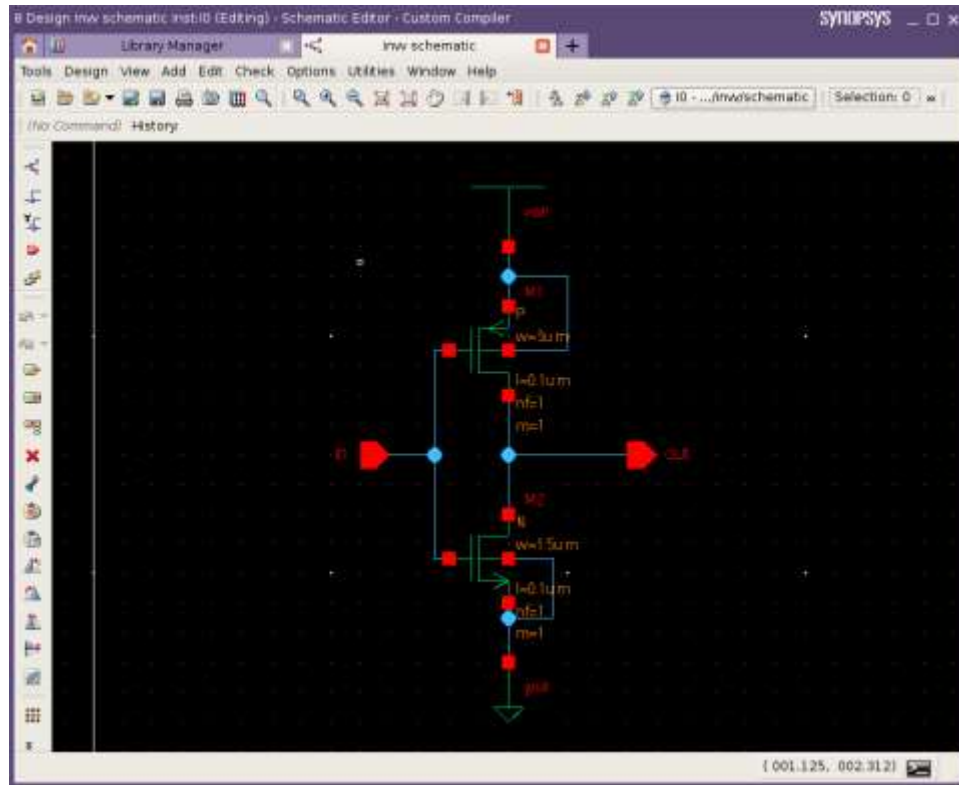
Bookmarks allow the designer a flexibility to easily navigate between cellViews within a session. Once the bookmark is created they can be referred at any time within that session.

In this task, you will learn to create and use Bookmarks to access design information.

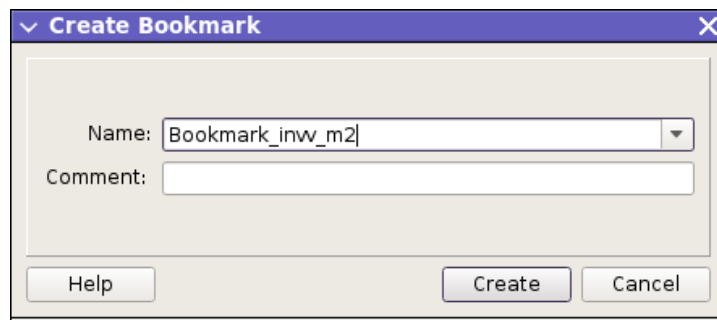
171. Descend two levels down the hierarchy of *VCO\_buff* to access the *invv* design as done in Task 4 using **Design > Hierarchy > Descend Edit** in the *VCO* schematic window (**Hint:** Use any instance of *VCO\_buff* and *invv* to descend)

**Note:** The schematic cellView *invv* is opened in the same window in edit mode.

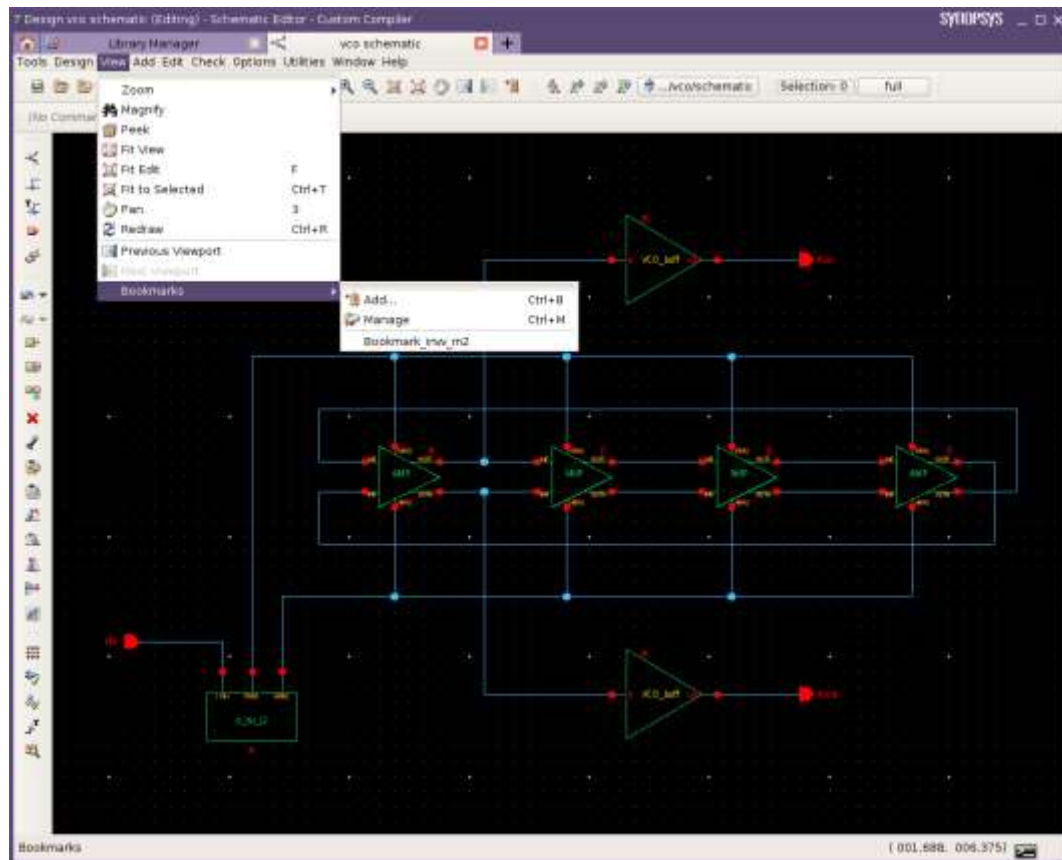




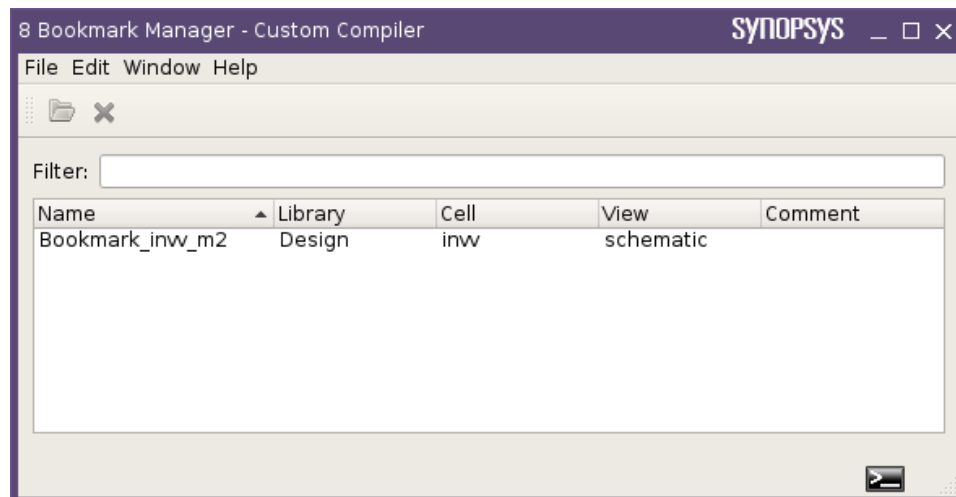
172. Zoom in to the instance *M2* using **View > Zoom > Area**.
173. Create a bookmark *book\_inw* using **View > Bookmarks > Add...**



174. Note down the parameter values of Width *w* and Length *l* for the nmos instance *M2*.
175. Return to the top level of the hierarchy that is, the *VCO* design using **Design > Hierarchy > Return to Top**.
176. You have successfully created a bookmark of the recently visited cell. If you wish to access the same cell later, you do not have to go all the way down the hierarchy again. You can refer to the bookmark that you have created.
177. Open the Bookmark Manager using **View > Bookmarks > Manage** from the *VCO* design.



**Note:** It opens up the Bookmark Manager where you can see all the information related to the created bookmark.



178. To access the bookmark, Right-click on the Bookmark\_inwv\_M2 entry in the Bookmark Manager and choose **Open Bookmark**.

**Note:** This will open up the bookmark in a new Schematic Editor window.

**Question 22.** What did you see when the bookmark was opened in the separate schematic window?

**179.** Close all schematic windows.

***Congratulations!***

You have successfully completed the lab!

# Answers / Solutions

## Task 8. Traversing the Hierarchy

**Question 1.** What is the other method which quickly takes you one level down the hierarchy without accessing command from the menu?

**180.** Double-click on the instance which you want to descend will quickly takes you one level down the hierarchy. It opens the descended view in an edit mode.

**Question 2.** How you can descend down the hierarchy in read mode?

**181.** The hierarchy can be descended in read mode using **Design** ⌘ **Hierarchy** ⌘ **Descend Read**. It opens the descended view in read-only mode.

## Task 9. Navigating the Hierarchy using Hierarchy Navigator

**Question 3.** What do you understand from *I5 (Design/VCO\_buff/schematic)*?

**182.** Here *I5* refers to the instance name of *VCO\_buff* cell placed in the VCO schematic. *Design* refers to the library name, *VCO\_buff* refers to the cell name and *schematic* refers to the view name.

## Task 10. Modifying symbol representation using Edit in Place

**Question 4.** Can you find out which are the instances you can edit?

**183.** You can only edit the instance *I6* of *VCO\_buff*.

## Task 7. Using Bookmarks

**Question 5.** What did you see when the bookmark was opened in a separate Schematic Editor window?

**184.** It opens up the view that was there at the time of the bookmark creation. In this case it opens the schematic cellView *invv*.

## 5. Hierarchy Configuration View Creation

### Learning Objectives

The purpose of this lab is to familiarize you with the Configuration View creation using the Hierarchy Editor and allow you to learn various selection rules and bindings by configuring the hierarchy for the post-layout simulation of a VCO design.

After completing this lab, you should be able to:

- Create and Edit the Configuration View
- Define and use the various selection rules and bindings

**Lab Duration:**  
**30 minutes**

# Introduction

A configuration is a set of rules that define which cellViews under a top-level cell are to be considered as part of the design for the netlisting, simulation, design partitioning for the mixed signal simulation and analysis.

During this lab, you will create a configuration view of the VCO testbench and will use the selection rules and bindings to configure it for the post-layout simulation of the VCO design.

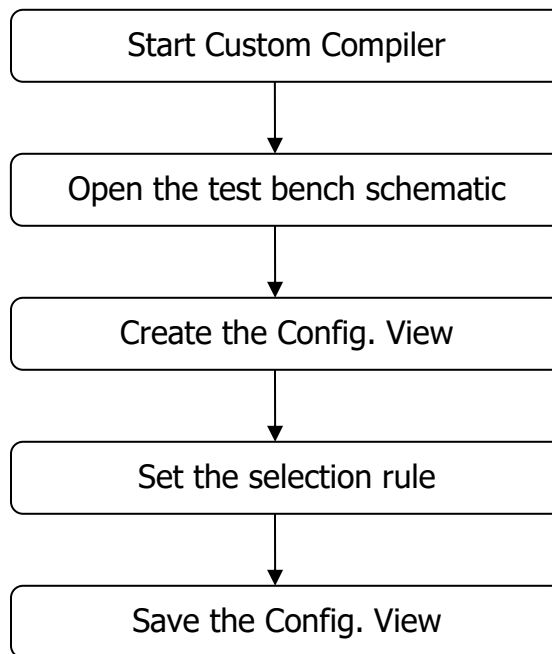
In this lab, you are provided with *HE* OpenAccess database library and a *lib.defs* file. The *HE* library contains the top *VCO* design, lower level cells and the testbench *VCO\_test*. You will create the *config* view of this testbench to do the post-layout simulation. The *lib.defs* is the default library definitions file that contains library name mapping to the physical location.

In order to create the configuration, view you need to understand the basic usage of the required commands.

Please see the section “Hierarchy Editor” of the *SE command reference manual* for the commands that are used in this lab.

# Flow Overview

## Lab 1 Tasks



# File Locations

All files for this lab are located in the directory  
*SE\_Hierarchy\_Configuration\_Lab1*

## Directory Structure

|  |                           |
|--|---------------------------|
| <i>SE_Hierarchy_Configuration_Lab1</i> | Current working directory |
| HE                                     | OpenAccess Design library |
| lib.defs                               | Library Definition file   |

## Relevant Files

|        |                 |
|--------|-----------------|
| ../PDK | Reference40 PDK |
|--------|-----------------|

## Answers & Solutions

There is an *ANSWERS / SOLUTIONS* section at the end of this lab. You are **encouraged** to refer to this section often to verify your answers, or to obtain help with the execution of some steps.



## Tool Versions

HSPICE

O-2018.09

WaveView

O-2018.09

# Instructions

## Task 53. Start Custom Compiler

---

185. In the Unix terminal window change your current working directory to *SE\_Hierarchy\_Configuration\_Lab1*. This will be your working directory for this lab.
186. Start Custom Compiler from the Unix prompt.

```
custom_compiler &
```

## Task 54. Open the Testbench Schematic

---

In this task, you will learn how to use a single testbench for doing multi-type simulation (schematic, layout and mixed-signal) using configuration views.

187. From the design library *HE*, open the schematic cell *VCO\_test* of the view name *schematic*.

**Note:** The *VCO\_test* cellView will be opened in a Schematic Editor window.

You will observe that the schematic contains two *VCO* symbols, one is connected to an output pin called *out\_schematic* and the other is connected to *out\_layout*. The idea of having two *VCO* in the schematic is to compare the output of the 'schematic *VCO*' to that of 'layout *VCO*' by creating and simulating the configuration view of this testbench.

188. Close the schematic cell *VCO\_test*.

## Task 55. Create the Config View

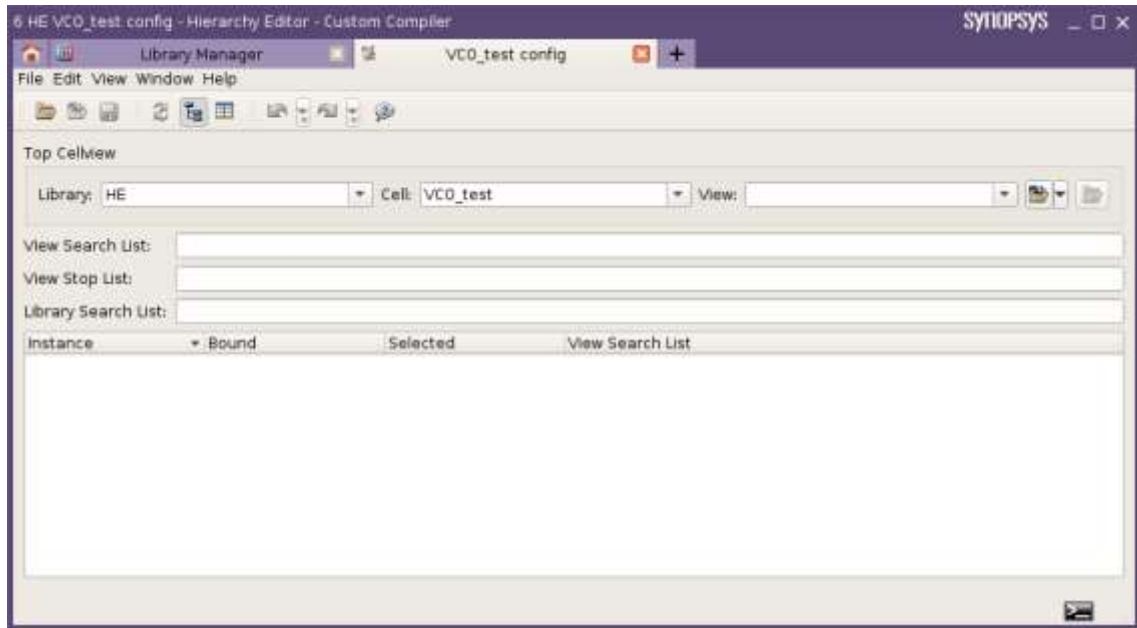
---

Once the basic testbench schematic is ready, Custom Compiler gives you the flexibility to verify your design at different levels of abstraction by creating the configuration view of this testbench.

In this task, you will create the config view of the *VCO* testbench for doing the post-layout simulation which accounts for the parasitic elements extracted from the layout.

189. From the Library Manager, create a config view of the cell *VCO\_test* with the view name *config* in the design library *HE*. (Make sure that in the **New CellView** dialog **Editor** is set to *Hierarchy Editor*)

**Note:** This brings up the Hierarchy Editor window. The **Top Cellview** field will be automatically filled as **Library: HE Cell: *VCO\_test***



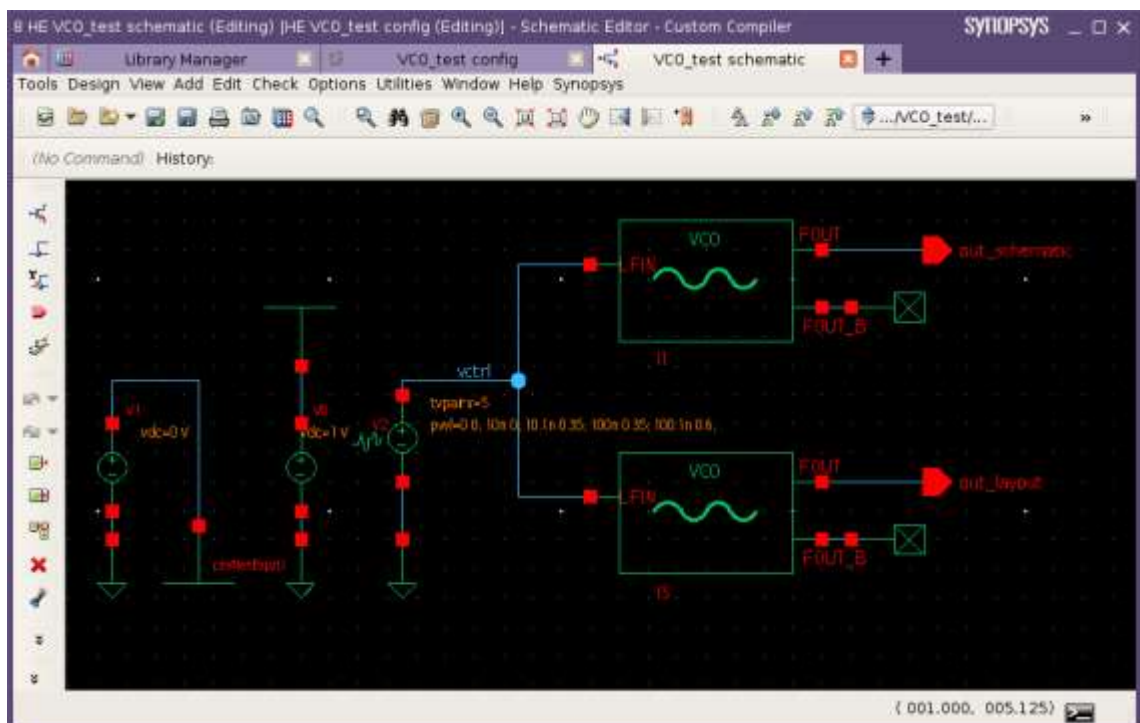
190. In the **Top Cellview** field, fill the **View** as *schematic*.

**Note:** After filling the View field, the “Open Configured Design” button, located in the right of that View tab, becomes active.



191. Click on the Open Configured Design button.

**Note:** This opens up the *VCO\_test* design in a Schematic Editor window in context with the Configuration view.



192. In the config schematic window, Double-click on any of the *VCO* instance to descend down the hierarchy. Check out the Console for any message.

**Note:** This event invokes the **Descend Into** command dialog.

**Question 17.** What do you interpret with **Descend Into** command dialog?

.....

193. Click on Cancel to close the command dialog.

## **Task 56. Setup the Selection Rule**

---

Once the config view is created, the next step is to configure the design using the Hierarchy Editor. Configure refers to the rules that define which cellViews are to be used for the placed instances under the top cell.

In this task, you will configure the design for the post-layout simulation and also learn the use of various selection rules and bindings.

194. Invoke Hierarchy Editor window by **Tools > Hierarchy Editor** command from schematic window.

195. In the Hierarchy Editor window, fill the **View Search List** fields with following views "*schematic hspice symbol*" and fill the **View Stop List** field with "*symbol*" view.

**Question 18.** What is the purpose of **View Search List** and **View Stop List**?

.....

**Question 19.** What do you see as you enter the View Search and View Stop List?

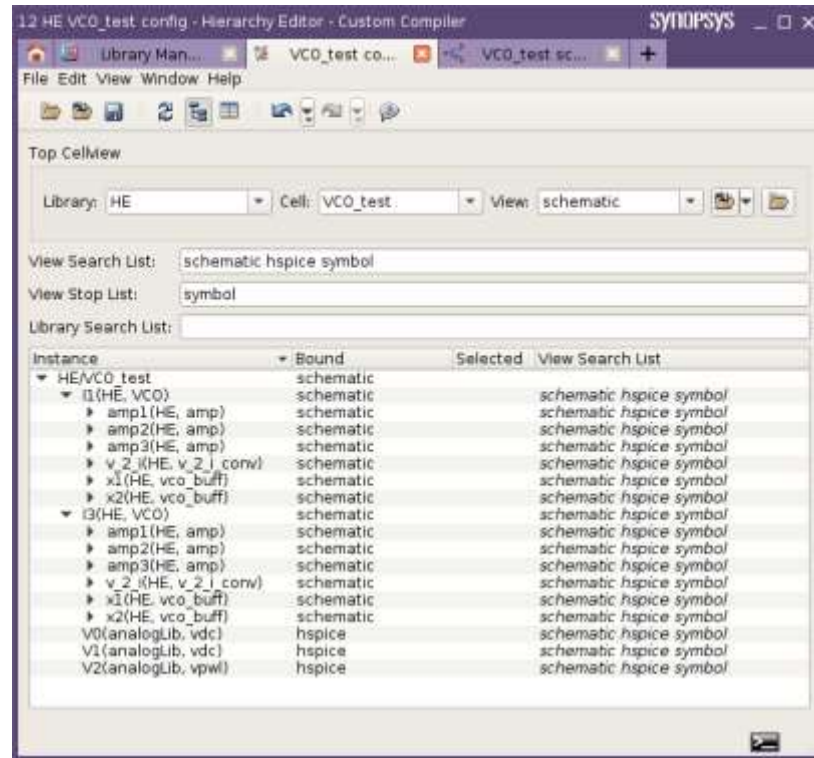
.....


196. Once the global bindings (**View Search List** and **View Stop List**) are specified, the information about the different instances in the testbench circuit can be seen in Hierarchy Editor.

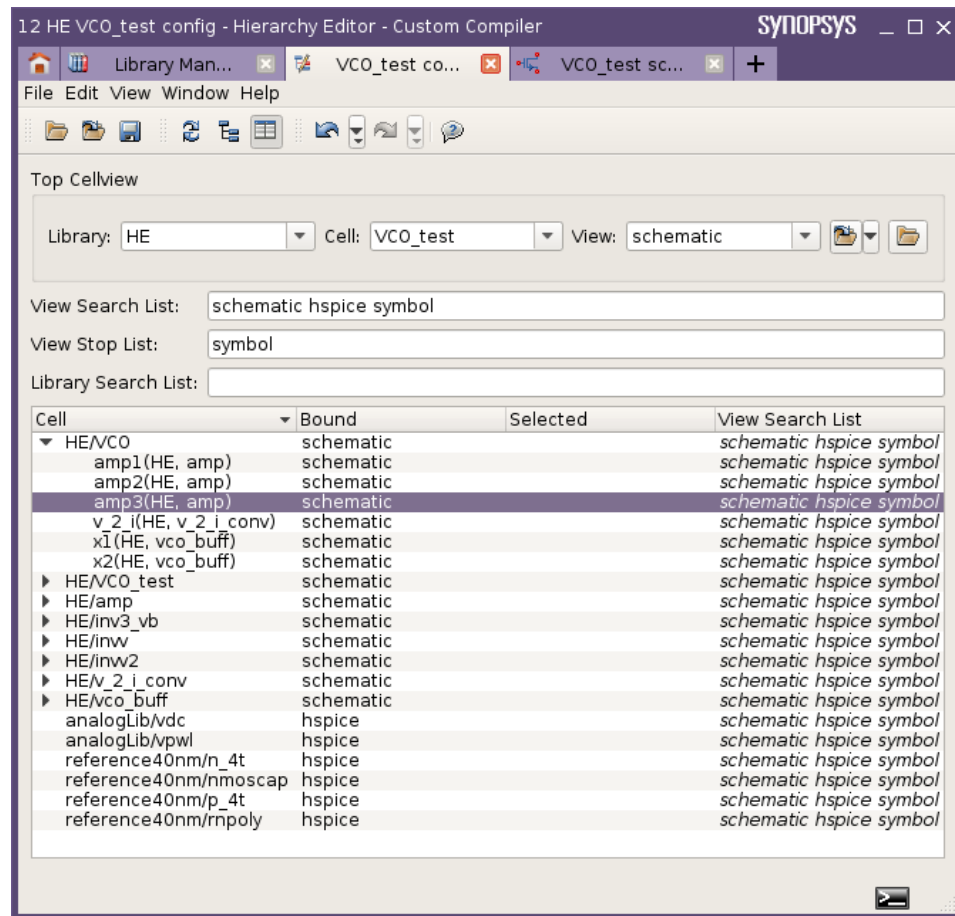
**Note:** By default, Hierarchy Editor shows the Tree Configuration pane displaying the design hierarchy in an instance tree. It allows you to modify the Instance and Occurrence level rules. Expand the instance tree as appropriate. You can see two *VCO*s, each having a unique instance number as placed in the schematic.

**Question 20.** What are the views found for each *VCO* in the **Bound** column of the Tree Configuration pane?

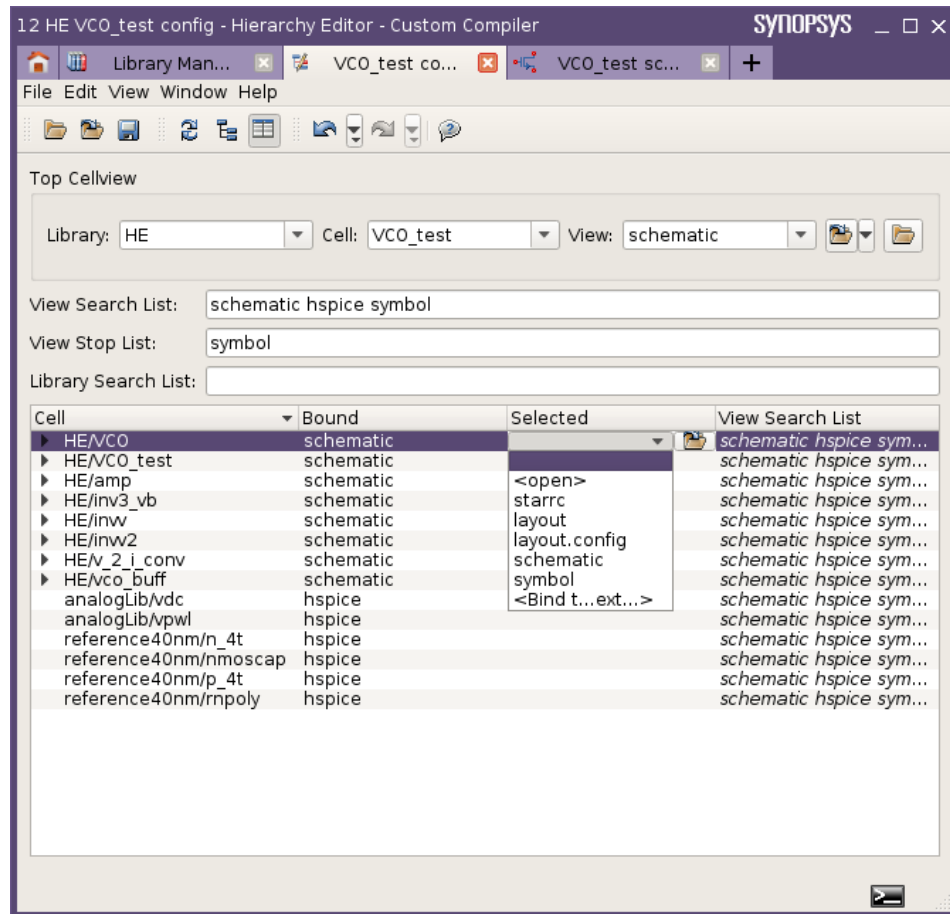
197. Expand the instance tree of *I1* (*HE*, *VCO*) and *I3* (*HE*, *VCO*) to see what views are bound to the lower level instances for each *VCO*.



198. Click the Save  button to save the config view.
199. Change the configuration pane to Cell in the Hierarchy Editor using **View > Cell**.



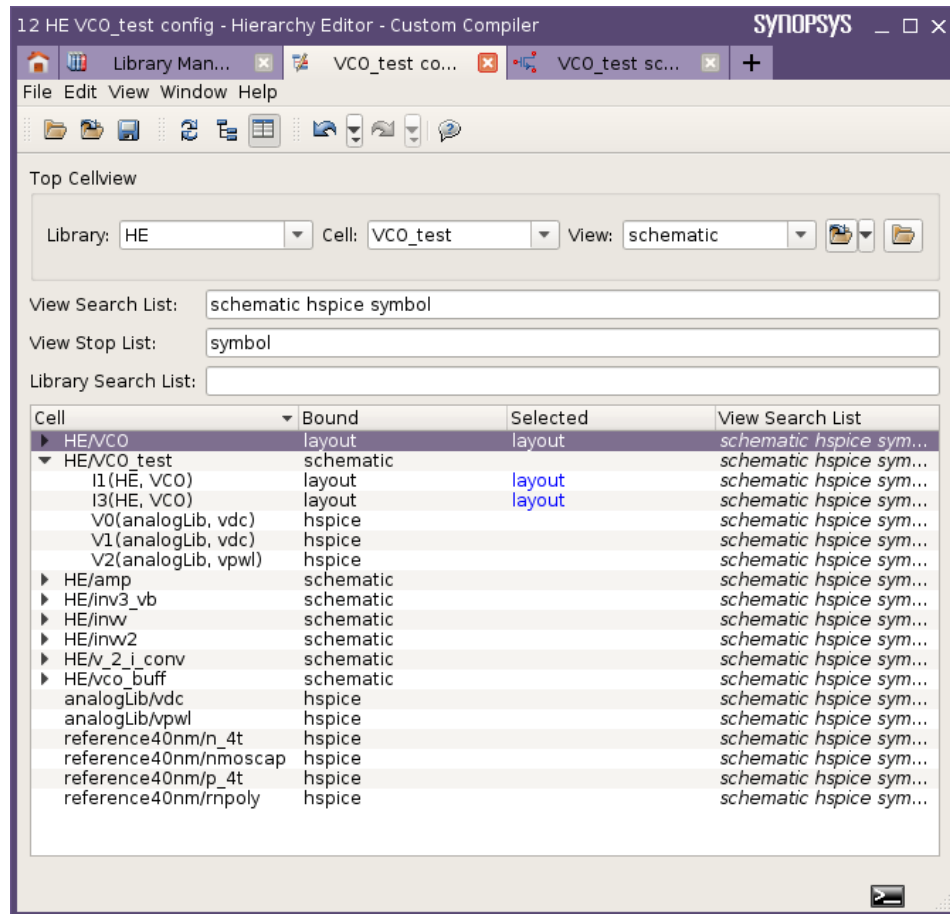
200. Click on the **Selected** column of *HE/VCO* cell entry as shown:



- 201.** Set the Cell view selection rule for *HE/VCO* cell to *layout* in the cell pane by setting the view **Selected** to *layout*.

**Note:** All the instances of *VCO* cell have now changed to view *layout*. Observe the entries of instances *X0* and *X1* under *HE/VCO\_test*.

The configuration view and the design automatically get updated as **Auto Refresh Design** and **Auto Refresh Config** under **View** menu of Hierarchy Editor is enabled by default.



202. In the schematic window, Double-click on the instance *I1* to descend into the hierarchy.

**Note:** As the switch view of instance *I1* has changed to *layout*, it descends into the *layout* view.

203. Return back to the top level using **Design > Hierarchy > Return to Top**.

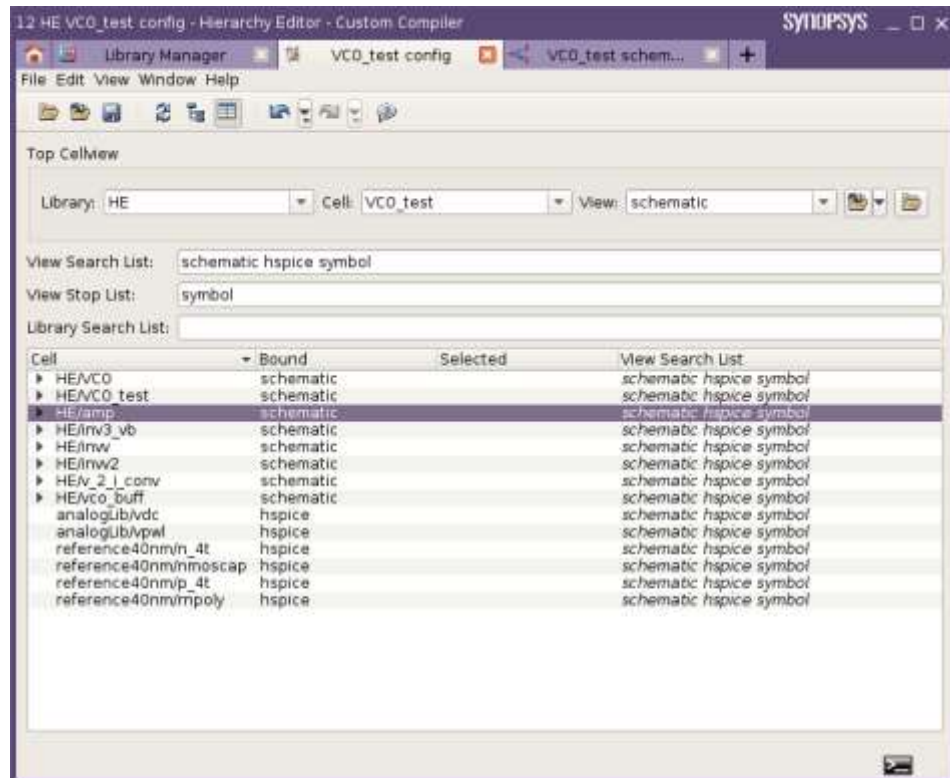
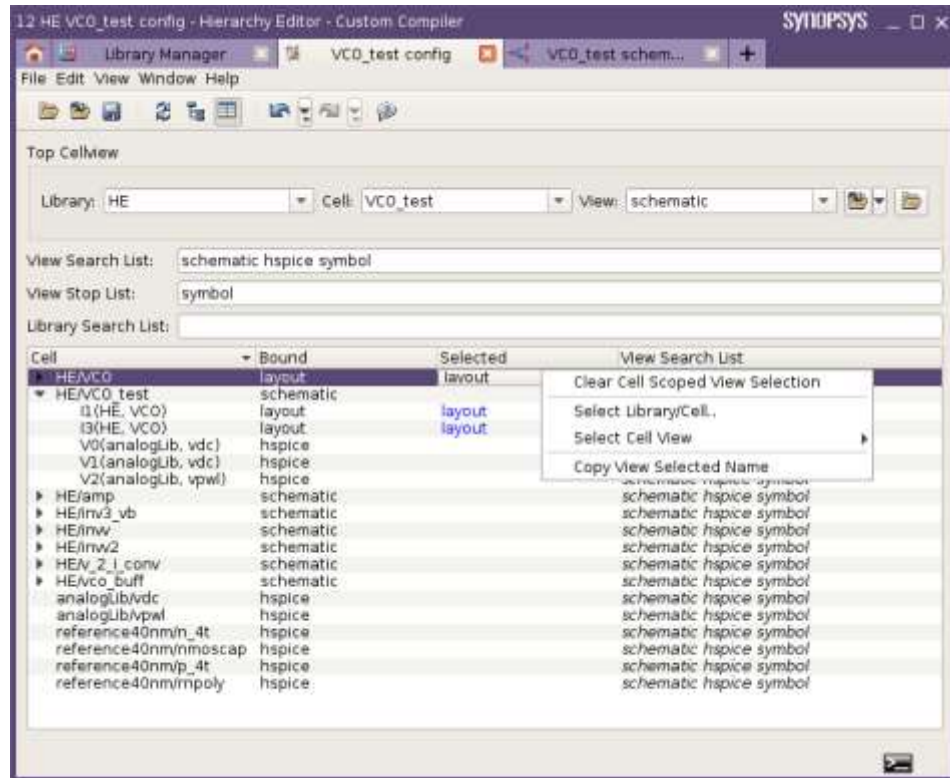
204. Similarly, try to descend into instance *I3* and return to the top.

205. You have learnt how to define the Cell view selection rule to configure your hierarchy. In the next step you will learn how to remove the selection rule.

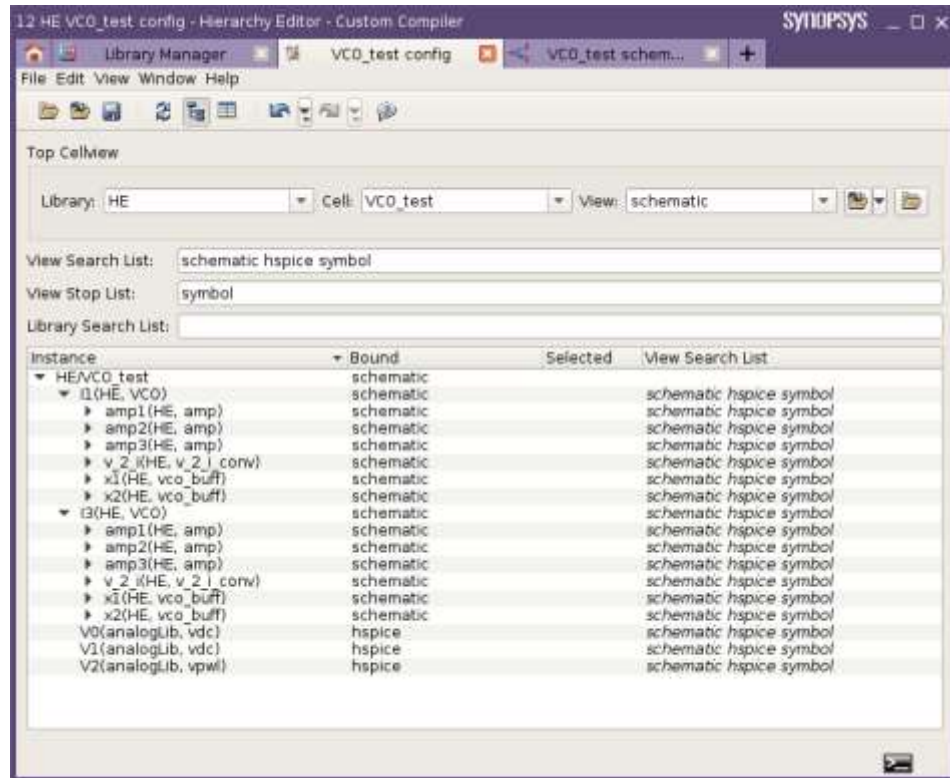
206. Remove the Cell view selection rule from *HE/VCO* cell using following steps:

- Place the mouse cursor over the **Selected** column of *HE/VCO* entry.
- Right-click to bring up the menu and choose **Clear Cell Scoped View Selection**.





207. Change the configuration pane to Tree in the Hierarchy Editor using **View > Tree**.



208. Now configure your hierarchy to carry out the post-layout simulation using Instance view selection rules.

209. In order to carry out the post-layout simulation you need to have the *starrc* view (parasitic extracted view) of the *VCO* design.

**Note:** *starrc* contains the parasitic elements, extracted from the layout, after the parasitic extraction process.

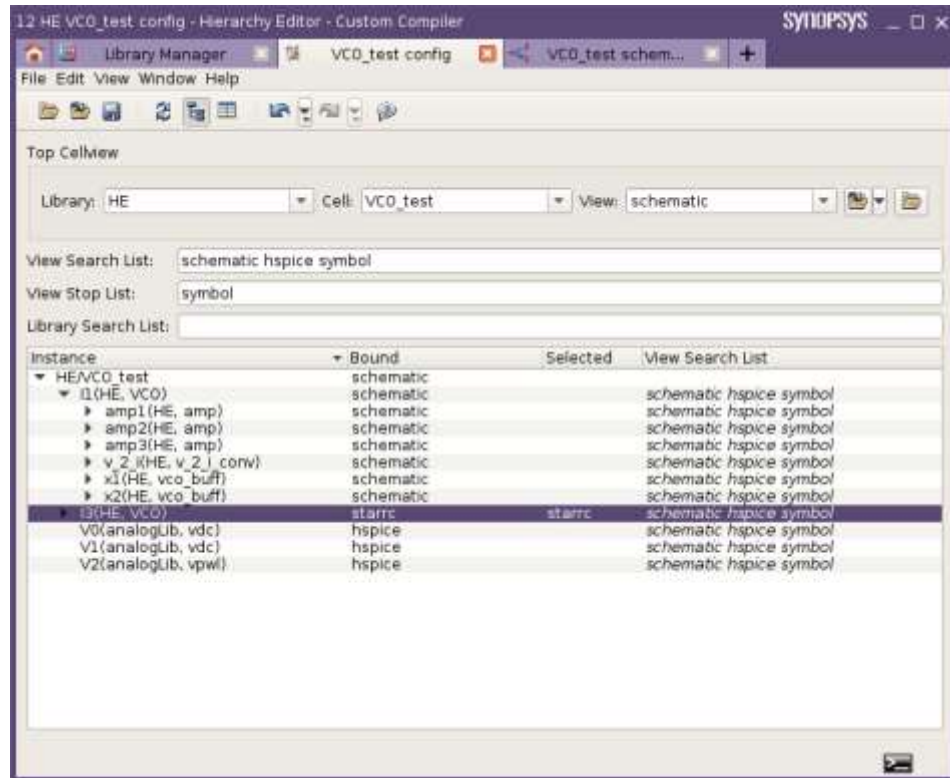
**Question 21.** List out the different views of the *VCO* design from the *HE* library.

---

210. Configure the selection rule of the *VCO* instance I3 (which is connected to the output pin *out\_layout*) in the Tree pane by setting the view **Selected** to *starrc*.

**Note:** The Instance view selection in the **Selected** column overrides the default view selection *schematic* for the *VCO* instance as shown in **Bound** column.

The configuration view and the design automatically get updated as **Auto Refresh Design** and **Auto Refresh Config** under the **View** menu of Hierarchy Editor is enabled by default.



211. In Hierarchy Editor, expand the hierarchy of instance *I3(HE, VCO)* and observe all the parasitic instances present in the *starrc* view (parasitic extracted view)
212. Save the configuration view using **File > Save**.
213. In the config schematic window, Double-click on the *VCO* instance *I3* to descend down the hierarchy.

**Question 22.** Which view do you descend into and how is it related to the above steps?

- 
214. Return to the top level of the hierarchy using **Design > Hierarchy > Return to Top**.

## **Task 57. Netlist and Simulate the Design using SAE**

At this stage, you have successfully created the config view. The next step in the design cycle is to netlist and run simulation for the config view.

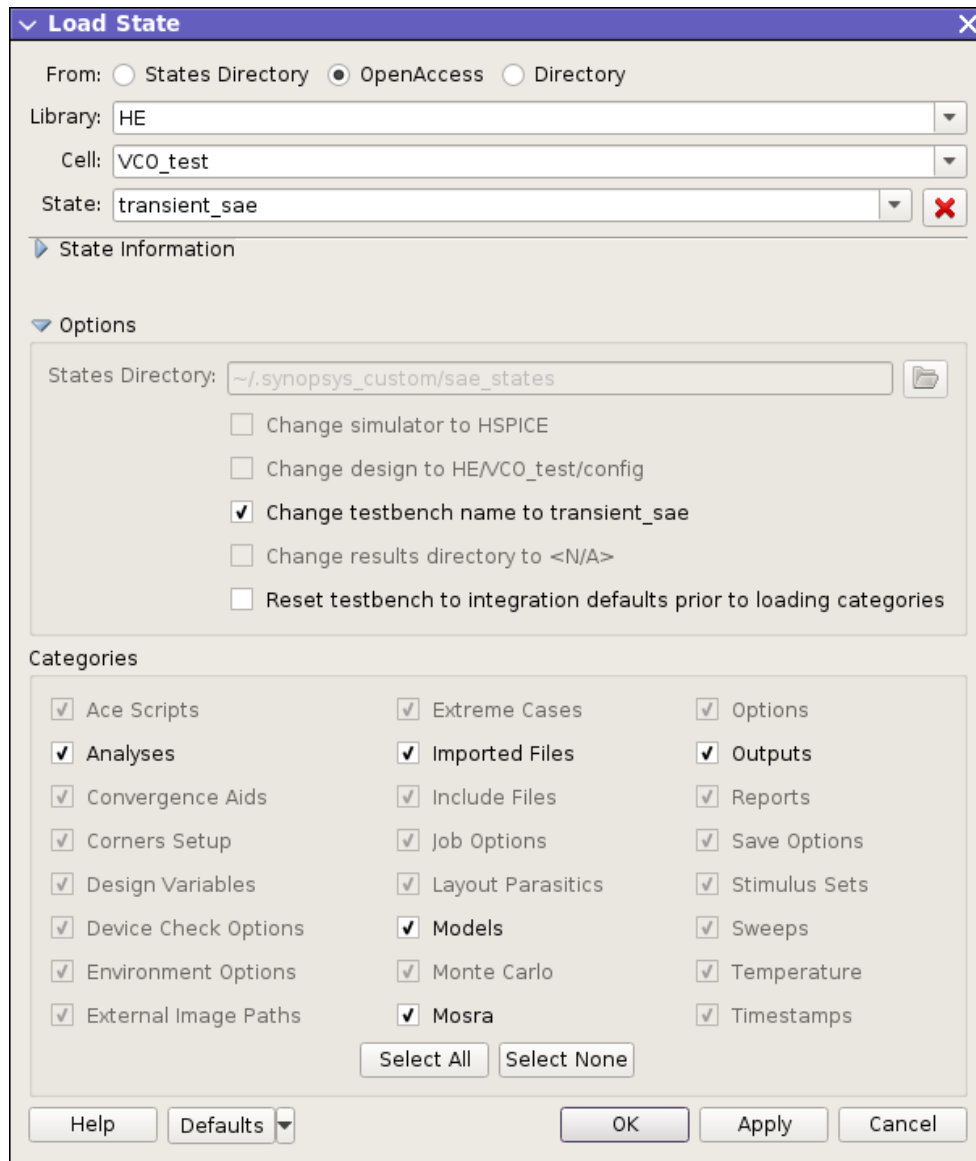
In this task, you will netlist and run the config view of *VCO\_test* which you have created and configured in Task 3 and Task 4.

You will be using Custom Compiler Simulation and Analyses Environment (SAE). For more information about SAE please refer to SAE training modules.

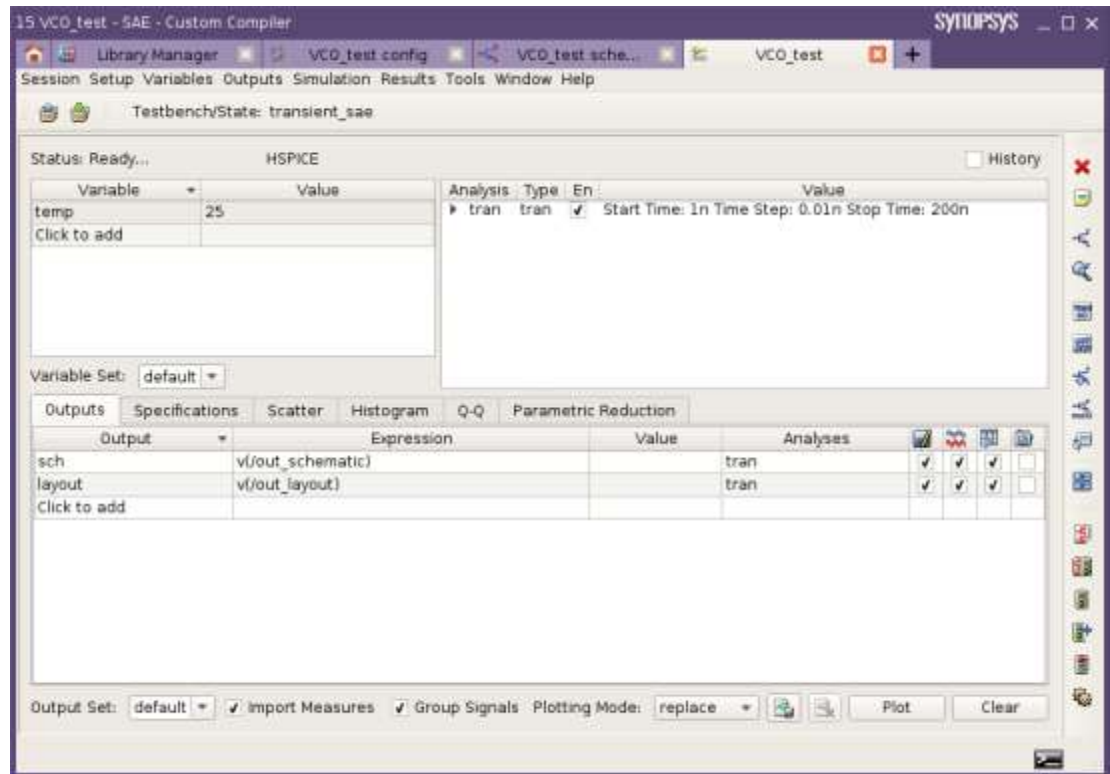
215. From the config schematic window, invoke the SAE using **Tools > SAE**.

216. From SAE main window use **Session > Load State...(L)**

217. Set **From** to *OpenAccess* and specify **HE/VCO\_test/transient\_sae** SAE state.



218. Press **OK** button to load the state.

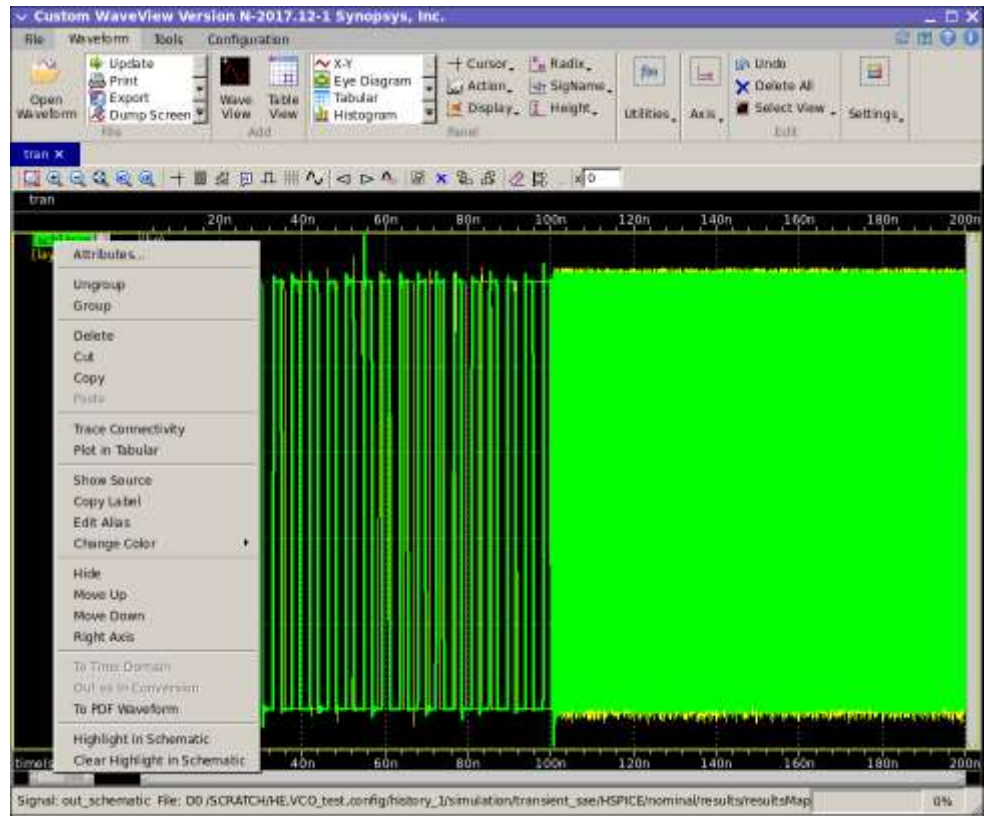


219. Netlist and run simulation using **Simulation > Netlist and Run (Ctrl-R)**
220. After simulation is completed, transient response of both the output signals (pre and post layout) will be plotted in Custom WaveView.

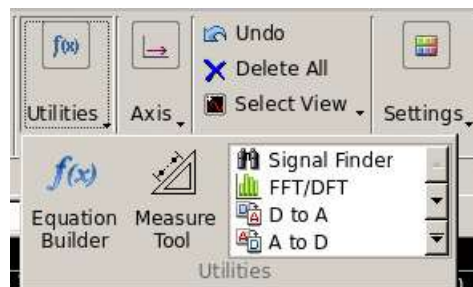
## **Task 58. Comparing the Waveform and Results**

In this task, you will measure the waveform and compare the results of layout to that of schematic.

221. Right click on signal name in WaveView to bring up Context Sensitive Menu and ungroup signals by clicking on “Ungroup” item

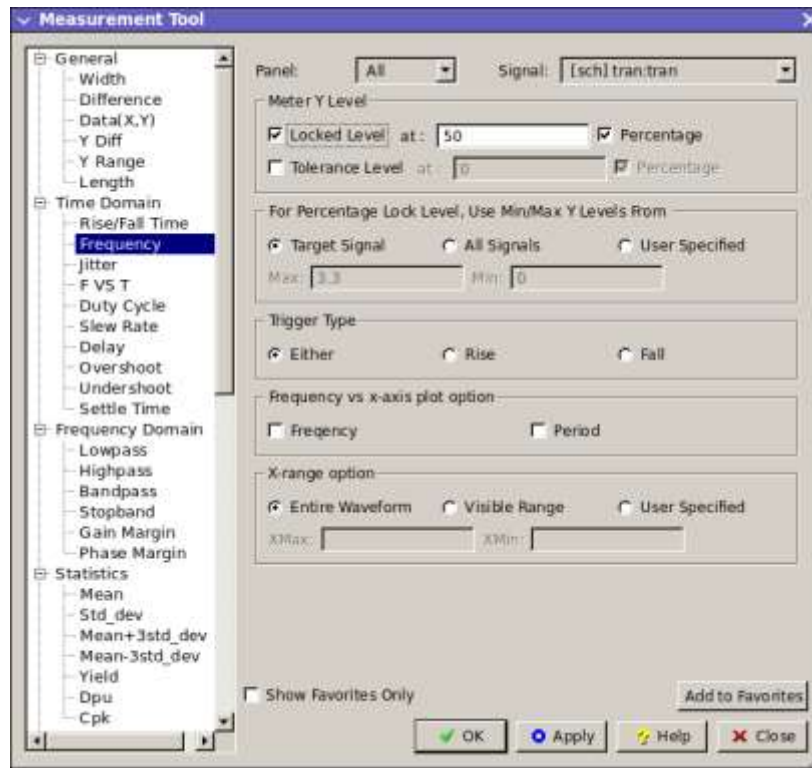



222. Measure the frequency of the transient waveform for both the outputs using **Utilities > Measure Tool ....**



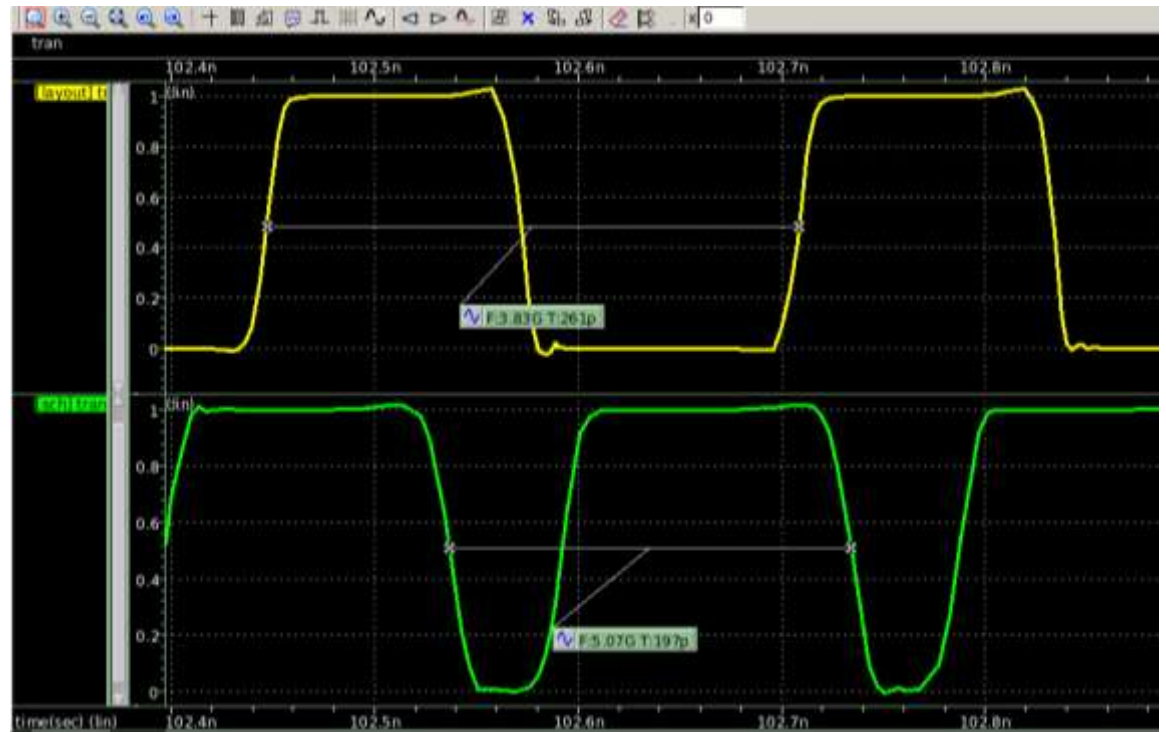
**Hint:** Use *Frequency* measurement and set **Meter Y Level** to *Locked Level* at 50.00 Percentage





**Note:** The D- meter  will appear. Move this D-meter on to the waveform on which you want to do the measurement. It will automatically snap to the specified **Meter Y Level**.

223. Note down the measured frequency value for *sch* and *layout* signals and see how much deviation you are getting after the post-layout simulation.
224. The measurement will look like this:



***Congratulations!***

You have successfully created the configuration view using Hierarchy Editor!!!!



# Answers / Solutions

## Task 8. Create the Config View

**Question 6.** What do you interpret with **Descend Into** command dialog?

- 225.** As there is no rule defined for instances present in the top cellView, on double-clicking on the instance the tool shows up the **Descend Into** dialog which shows the available views which user can descend into.

## Task 9. Setup the Selection Rule

**Question 7.** What is the purpose of **View Search List** and **View Stop List**?

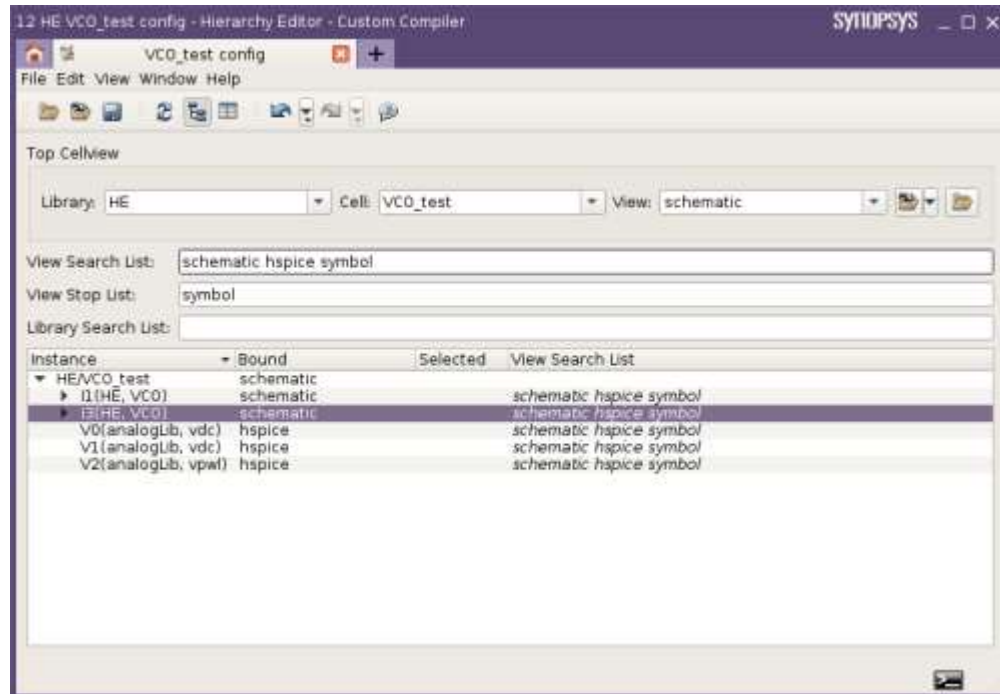
- 226.** The **View Search List** is an ordered list of view names used to determine the switched master of an object. The algorithm used to determine the view to use for an object is: scan the View Search list from left to right. If the view exists for the object's cell then the view is bound to the object. If none of the views in the view search list exist then no view is bound to the object.

The **View Stop List** provides a mechanism for stopping recursive descend into design hierarchy. Note that View Stop Lists are optional and are not required to be specified by the user

**Question 8.** What do you see as you enter the View Search and View Stop List?

- 227.** The *HE/VCO\_test* entry will appear in the Tree pane of the Hierarchy Editor window.

## Lab 1



**Question 9.** What are the views found for each *VCO* in the **Bound** column of the Tree pane?

**228.** As the first entry in the **View Search List** is *schematic* and this view exists for the design. Therefore, the view **Bound** for both the *VCO* will be *schematic*.

**Question 10.** List out the different views of the *VCO* design from the *HE* library?

**229.** *schematic, layout, symbol, starrrc*

**Question 11.** Which view do you descend into and how is it related to the above steps?

**230.** As the instance binding set to this instance is *starrrc*, you will descend into the *starrrc* view of the *VCO* instance *I3*.

