

**Walkthrough 3
Full Custom Using Standard Cells**

Author Neil Cole
Date Thu 5-May-2011 10:27 am

1.1	Important Notes	3
2	Conventions used in this document	3
2.1	Entering Parameters for commands	4
3	Accessing and Configuring the Design Environment	4
3.1	Opening a Terminal Window on the Linux Platform	4
4	Creating a Design Environment	5
4.1	Using the Transport Form	6
5	Starting Cadence and the AMS 0.35um Design Kit	8
6	SCHEMATIC ENTRY	9
6.1	Introduction	9
6.2	Creating a schematic cellview called smotor	9
6.2.1	Supplementary Information	10
7	Creating the Schematic	11
7.1	Entering your Schematic Components	11
7.2	Peripheral padsSchematic Components	13
7.3	Inputs	14
7.3.1	Clock Input	15
7.3.2	Dir	17
7.3.3	Final	18
7.4	Outputs	18
7.4.1	Connecting Test1 outputs	18
7.5	Connecting BCDCOUNT outputs	19
7.5.1	Labelling the bus and single bit wires	20
7.5.1.1	Labelling the buses	20
7.5.1.2	UNIT	21
7.5.1.3	TEN	21
7.6	Connecting display_chip Outputs	23
7.7	Final Output Schematic Structure	23
8	Adding Power	24
9	Final Schematic Structure	25
10	Digital Simulation	26
10.1	Editing the Test Fixture	28
11	Generating a Full Chip Layout	30
11.1	Getting the required files.	30

11.2	Extracting the files	31
12	Configuring the Design Area : Now use the A4407Layout area	32
12.1	Transferring the Verilog File	33
12.2	Fixing and Editing the Verilog File global nets	33
12.2.1	Starting Encounter	34
12.3	Basic Configuration	36
12.3.1	Make the following changes to the form	39
12.3.1.1	Advanced Options	40
12.3.1.2	Saving our work on the configuration file	40
12.4	Implementing the changes	40
12.5	Generating and Using the IO file to control the location of peripheral cells..	41
12.5.1	Putting the Power Pads where we want them	41
12.5.2	Creating the IO Pads file	41
12.5.3	Setting up the IO pads	42
12.5.4	Editing the smotor.io.save file	42
12.5.5	Save the changes and Exit the editor	45
12.6	Including the changes in your Design Import configuration file	45
12.7	Using the new configuration file to create the design	45
12.8	Configuring the AMS scripts	46
12.8.1	Loading the AMS scripts	47
12.8.2	Routing our selected Power Supplies	49
12.9	Pre and Final Routing Operations	50
12.10	Final Actions	51
13	Caveats	51

1 Introduction

This document is intended to replace the Entry and Simulation of Full Custom Components in the current 4407. Whilst sub optimal in terms of course development the intention is to replicate the functions covered in the original walkthroughs.

During this exercise we will place the three previously created elements onto a schematic. Add the appropriate peripherals and power supplies and using the SOC Encounter tools produce a final chip layout.

Compared to the original walkthrough the change are:

Table 1:

Change	Original	Update
Design Kit	Mietec 2um	AMS 0.35
Simulator	cdsSpice	Spectre
Environment	Unix Direct	UNIX/Linux Transport
IC Design	1999-2000	4.5/4.6
Layout	Virtuoso	Layout XL
Router	LAS	Encounter / Assura

1.1 Important Notes

Due to limitations with software environments and licenses this work is only possible on the Linux system. Areas have been implemented for all students on the Linux platform.

Until an updated version of the walkthroughs are available you should use the local version of the "full custom cell" called "display_chip" and not your local version of "display"

2 Conventions used in this document

The following are conventions that will be used in this document.

- Where user input is required it will be delineated with "<>".
- Where a special function key such as escape, control or return/enter is indicated in a command it will also be enclosed in "<>" brackets. For example "<RETURN>".
- Please be aware of spaces in words and commands. Like Windows and common English UNIX and Linux require spaces between commands.

- Unix and Linux commands are lower case unless stated. Unix and Linux are case sensitive.
- Operations involving the mouse will use the left hand mouse button unless otherwise stated.
- Options selected from sub menus will be indicated in the following manner

Main Menu Item -> Sub menu Item -> Sub Menu Item

- e.g. File -> New -> Library

2.1 Entering Parameters for commands

Unlike windows most UNIX/Linux parameters are delineated by a "<SPACE>". For example

- `grep -i fred *.v`

3 Accessing and Configuring the Design Environment

3.1 Opening a Terminal Window on the Linux Platform

Linux is, unsurprisingly, very similar to UNIX. It has become the defacto "technical" platform replacing or supersceding UNIX systems. The environment of the Linux platform has been configured to be as similar to the UNIX environment as feasible.

There are two ways to open up a terminal window in Linux

1. Move the mouse cursor onto the background of the Linux display and use the right hand mouse button to display the context sensitive menu. Select the Terminal option from this menu.

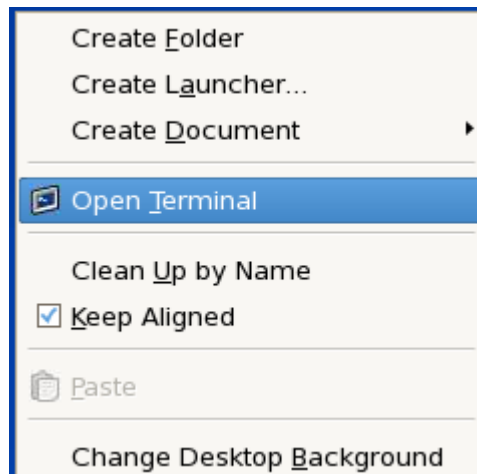


Fig: Background Context Sensitive Menu

2. Use the Application -> Accessories -> Terminal menu option on main menu.



Fig: Linux Top Level Terminal Menu

4 Creating a Design Environment

In order to support the current tool flow and avoid having to move students to version 6 of the Cadence tools, which are significantly different it has proven necessary to use two design areas. To ease students work these have been temporarily added to the transport tools.

The A4407Design Walkthroughs area is to be used for all Schematic and Simulation work

The A4407Layout Walkthroughs area is to only be used for the encounter tools.

The only common file that will be passed between the two environments will be the final.v file. Which the guidelines will tell you to copy to your ~ area.

Students using the AMI design system are provided with a Transport menu that facilitates easy access to appropriate design directories and associated applications software.

Accessing and Configuring the design environment is achieved using the “transport” tool. Students should start a UNIX terminal window and invoke the transport tool by typing the command:

- transport & <RETURN>

The *Transport* form should appear as shown below.



Fig: Transport Menu with no module areas created for A4407Schematic with Walkthrough selected

The menu enables a module design area to be created and then entered by specifying the module being studied. Modules are classified by type and function.

4.1 Using the Transport Form

Click on the **EDA** button to display a list of modules within that classification.

- Select the module **A4407Design**

If you are using the transport tool for the first time for this module there will be no design areas

Full Custom Using Standard Cells

present and the “Area” field will be blank. In this case you will need to create the default structure by using the “Create Module Area” button.

- Click on the **Create Module Area** button to the design areas.

This may take a few seconds. Once this has run successfully the “Area” field will be populated. As shown in the figure “Transport Menu with module areas specified”.

Now select the Walkthroughs area. Only the *Walkthroughs* area will be used for this exercise.

- Click on the **Walkthroughs** button

The *Transport* menu should now be as shown below.

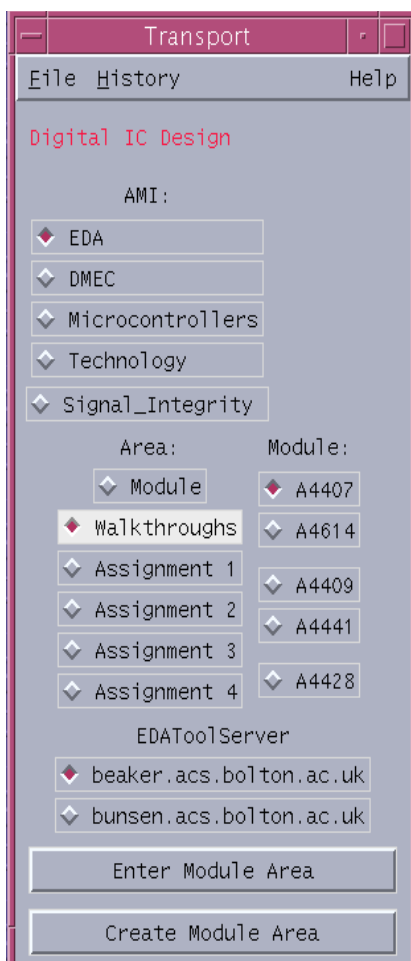


Fig: Transport Menu with module areas specified

Now click on the **Enter Module Area** button to open up a configured UNIX terminal window.

All subsequent commands will be typed in this window. This window will be in the correct location, in this case the “Walkthrough” area. For the 4407 module this will be physically located at “~/AMI/

A4407/Walkthroughs". This area is also configured to allow you to run all the relevant tools and

5 Starting Cadence and the AMS 0.35um Design Kit

In the configured 4407 terminal window you can start the design kit using the UNIX command:

- **amiselect fb <RETURN>**

After a short delay the CADENCE Command Interpreter Window (CIW) as shown below will appear at the bottom of the screen. The window provides a tool bar for the top level menu commands and a display window for status information.

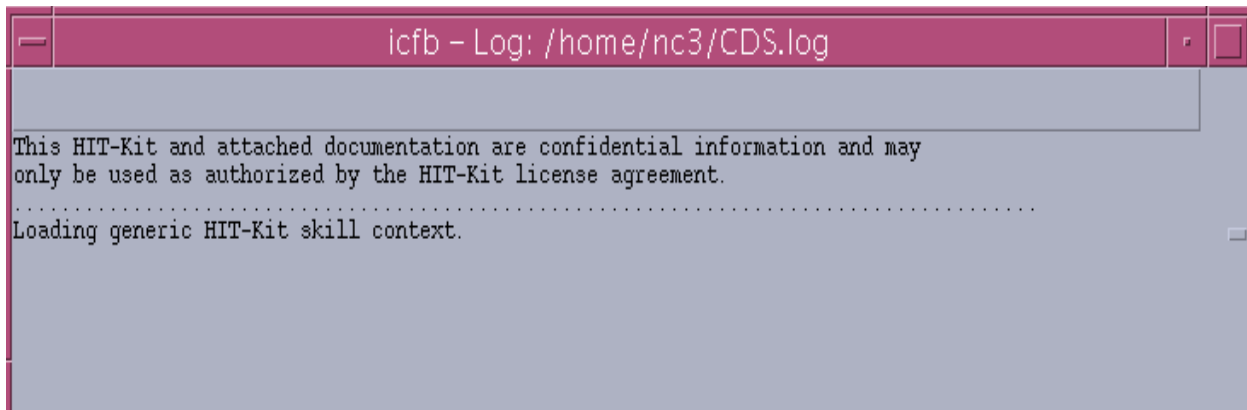


Fig: The CADENCE Command Interpreter Window (CIW)

The Cadence *Library Manager* as shown below will also be displayed.

Full Custom Using Standard Cells

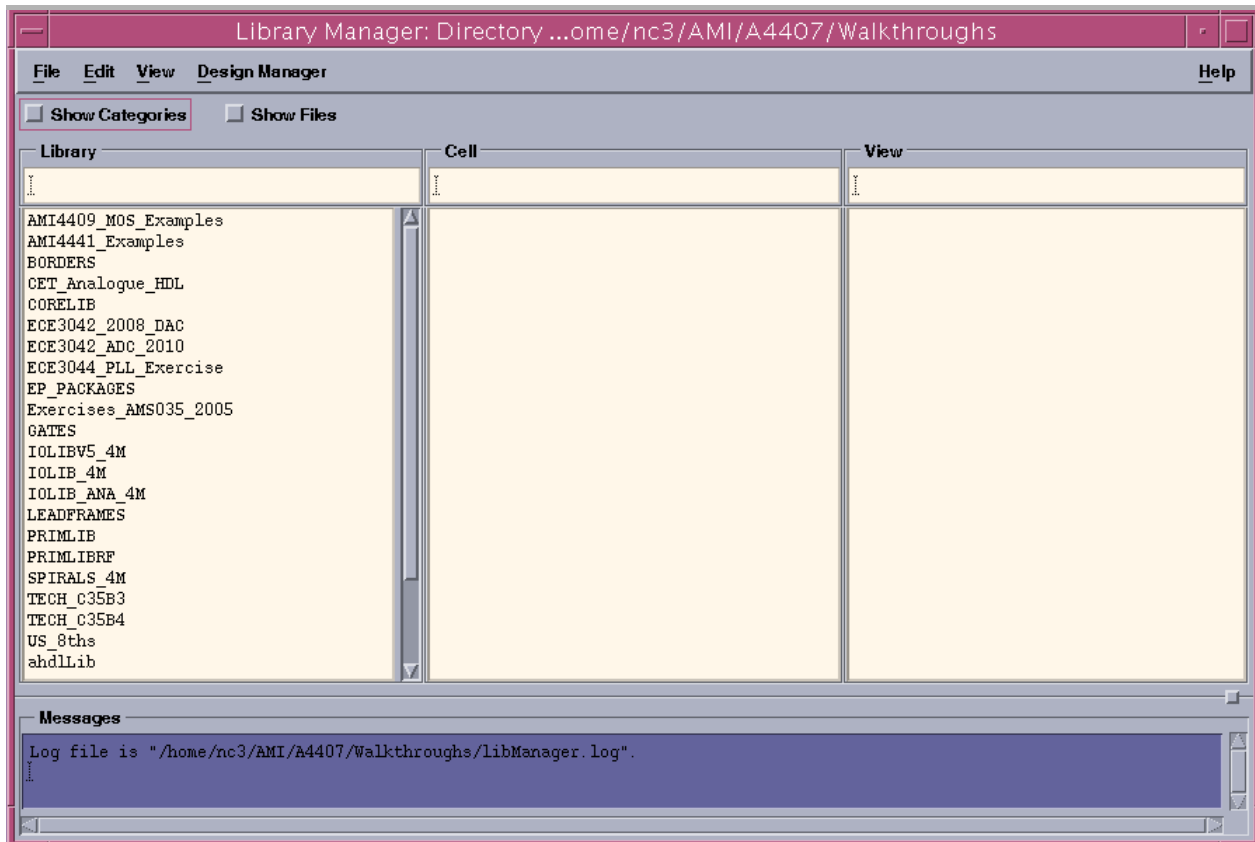


Fig: Library Manager

The *Library Manager* enables design libraries, cells and cell views to be created, opened, copied,

6 SCHEMATIC ENTRY

6.1 Introduction

This section details the operating instructions for invoking CADENCE and running *Composer* to enter the stepper motor driver schematic.

Instructions are provided for entering logic components, input/output pins and wires and for checking and saving the design.

6.2 Creating a schematic cellview called smotor

Click on the **examples** library in the *Library Manager* and verify that it highlights.

Select **File -> New -> Cellview** from the *Library Manager* to display the *Create New File* form.

Full Custom Using Standard Cells

Click in the *Library Name* box to display the list of available libraries.
Select **examples** from the list.

Click in the *Cell Name* box and enter **smotor** as the cell name.
Verify that the *View Name* box has been set to **schematic**.
Verify that the *Tool* box has been set to **Composer -Schematic**.

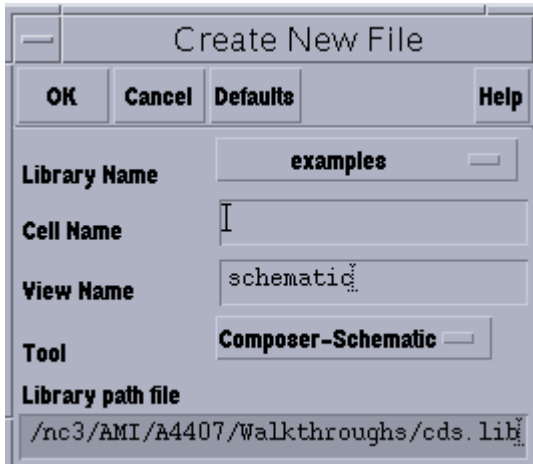


Fig: The Create New File form

Add in the entries for the cell name and accept the form by clicking on **OK**.

A message will appear in the CIW confirming successful creation of the cellview and an empty schematic window will now open.

6.2.1 Supplementary Information

Note: In subsequent sessions should you wish to display or edit the schematic that you are about to create you can select it either from the CIW or via the Library manager. The recommendation would be to use the Library manager

From the *Library Manager* select the required *Library Name* (**examples**), *Cell Name* (**test1**) and *Viewname* (**schematic**) as shown below

Select **File - Open** from the *Library Manager* to open the schematic, or right click on the schematic view to bring up the context sensitive options.

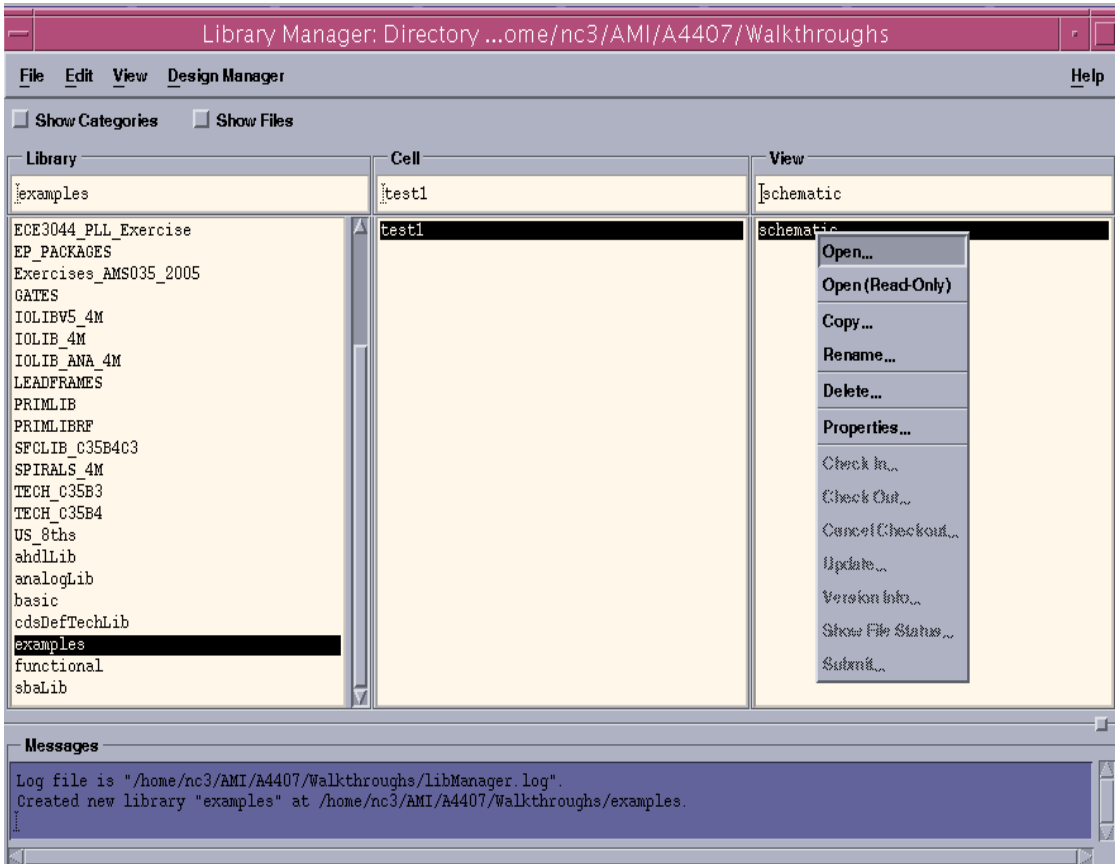


Fig: Selecting the test1 schematic from the library browser

7 Creating the Schematic

The schematic window displays the main options along the top and a subset of these options in icon form down the left hand side.

The circuit schematic shown below will be assembled in three stages:-

1. Entering and connecting the main components
2. Adding the pads and power pins
3. Connecting the pads and power pins

Fig: Final Schematic

7.1 Entering your Schematic Components

Ensure that the three elements we have created previously should all have symbols. If not create them following the route covered in the previous walkthroughs.

The ones provided are called:

Table 2: Schematic Library Components

Library	Name	View to use
examples	bdcoun	symbol
examples	display_chi p	symbol
examples	test1	symbol

Place these on the smotor schematic as shown.

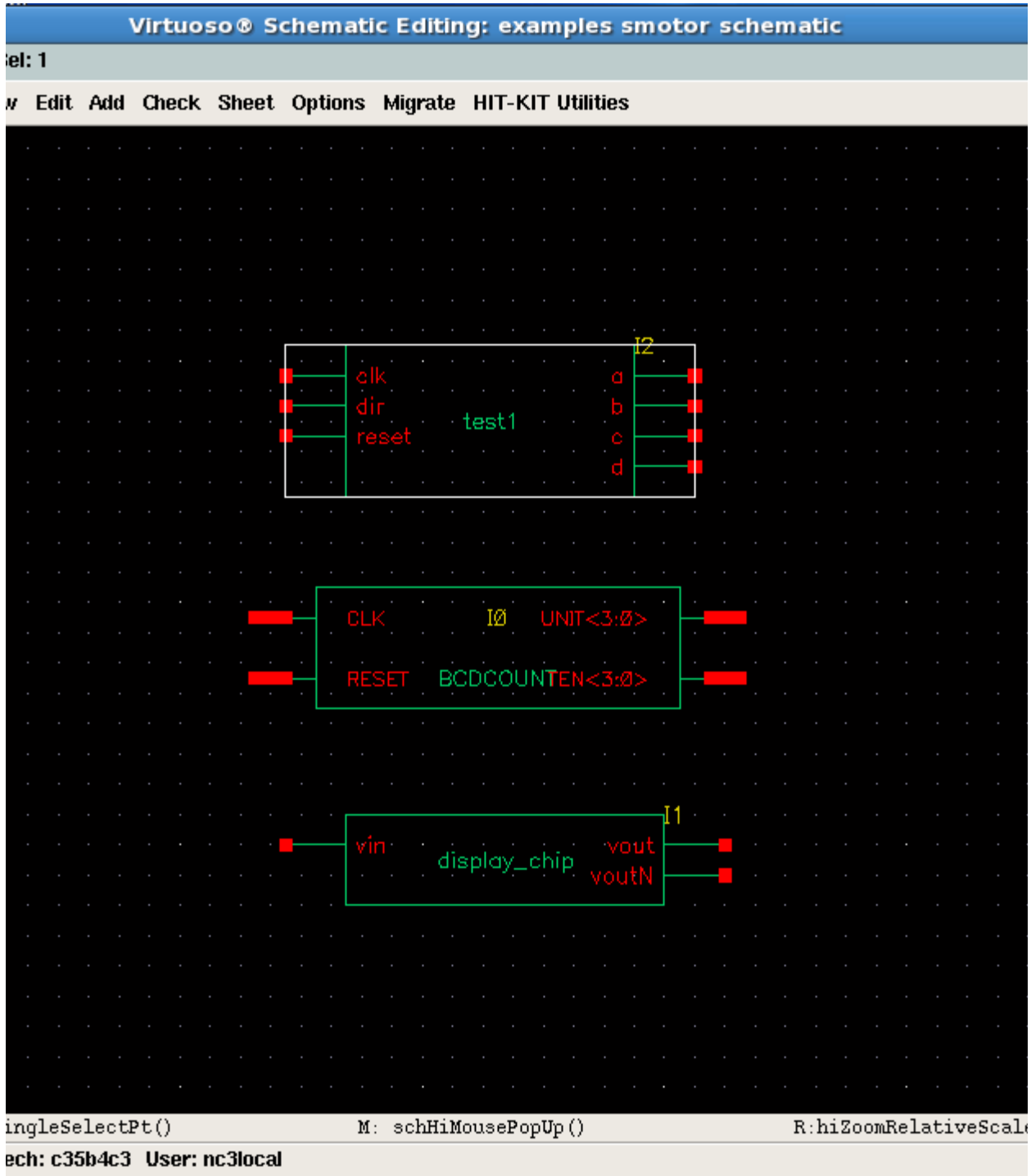


Fig: Initial smotor Placement

7.2 Peripheral padsSchematic Components

We now need to place the peripheral cells that will be used on the final chip. These will include Power, Ground, Input and Output cells.

We are going to use the peripheral cells from the IOLIB_4M. Which will be the 3.3 volt compatible peripheral cells.

Table 3: Schematic Library Components

Library	Name	View to use	Function
IOLIB_4M	ICP	symbol	CMOS Input
IOLIB_4M	BU2P	symbol	CMOS Output
IOLIB_4M	GND3ALL P	symbol	Power
IOLIB_4M	VDD3ALL P	symbol	Ground

Use the mouse to navigate “Library Browser - Add Instance” form and select the cell and cell view required.

7.3 Inputs

Place 3 instances of the IOLIB_4M ICP on the left hand side of the schematic.

Place schematic input pins on the left hand side of the ICP instances with the following names:

- Clock
- Reset
- Dir

Connect the pins to the ICP components.

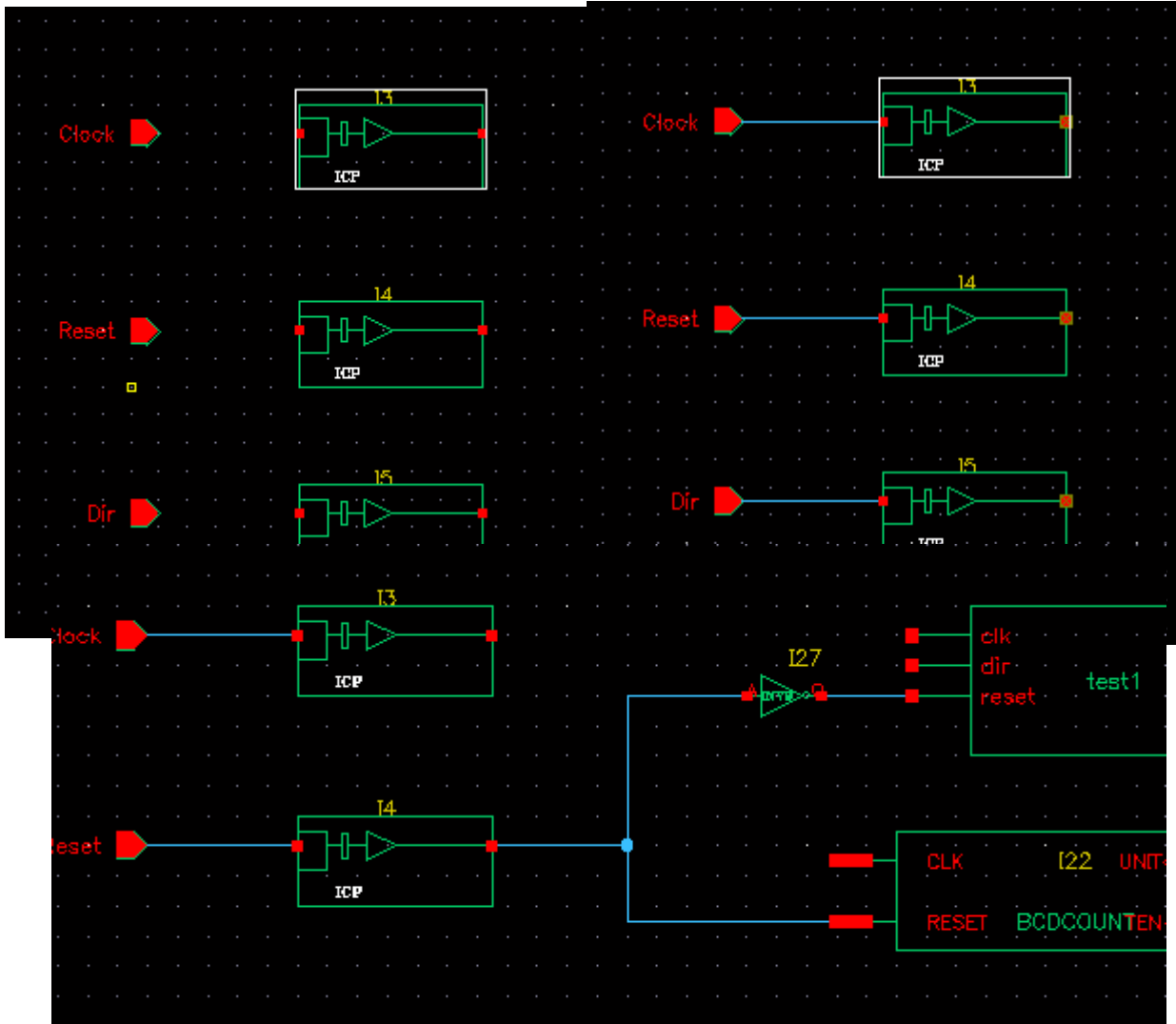


Fig: Figure placement and wiring of input pins

7.3.1 Clock Input

Wire the output of the ICP connected to the Clock pin to the clock inputs on the test1 and BCDCOUNTEN cells.

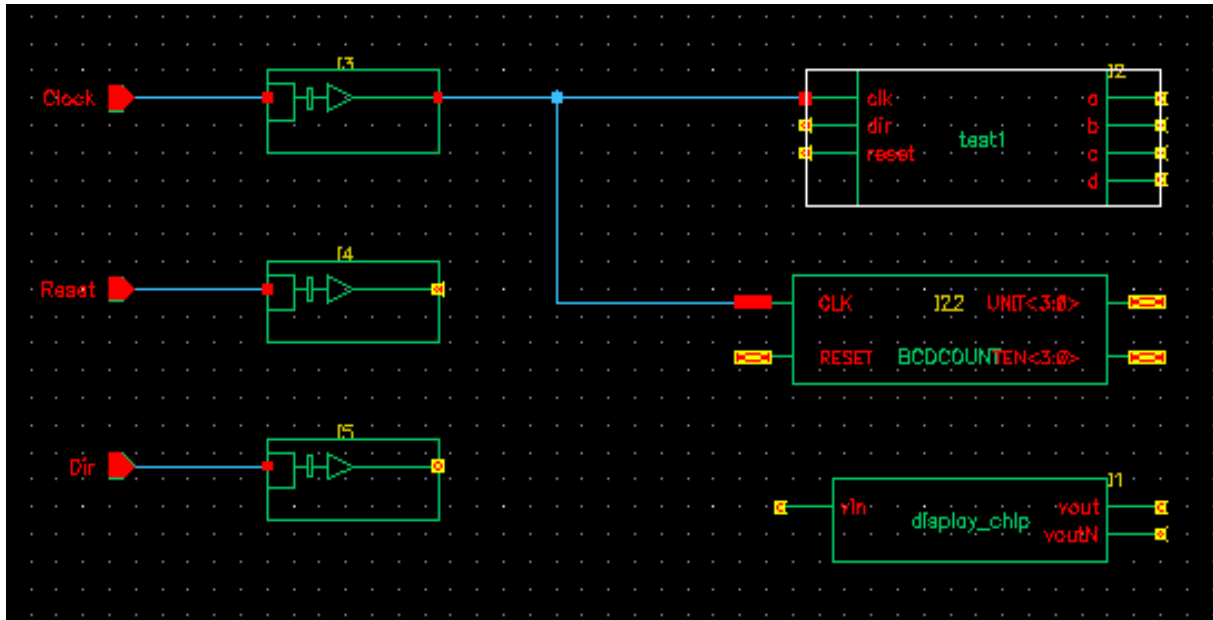


Fig: Schematic Clock Wiring

Reset

The Reset on the test1 and BCDCount inputs are different.

The test1 Reset is highlight active and the BCDCount one is low active so we need to invert the reset signal using an inverter.

Library	Name	View to use
CORELIB	INV0	symbol

Place this inverter instance between the Reset pin and the test1 instance.

Wire the output of the ICP connected to the Reset pin to the reset input and BCDCOUNT instances.

Fig: Schematic Reset Wiring

7.3.2 Dir

Wire the output of the ICP connected to the Dir pin to the dir input on the test1 instance, and the vin pin on the display_chip instance.

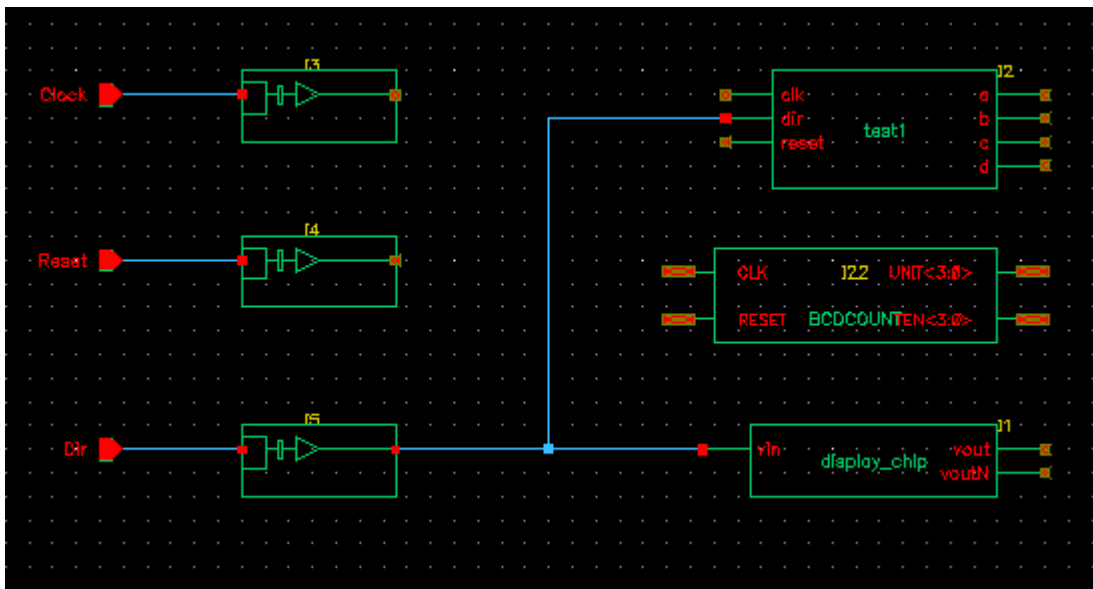


Fig: Schematic Dir Wiring

7.3.3 Final

The final input structure should resemble this:

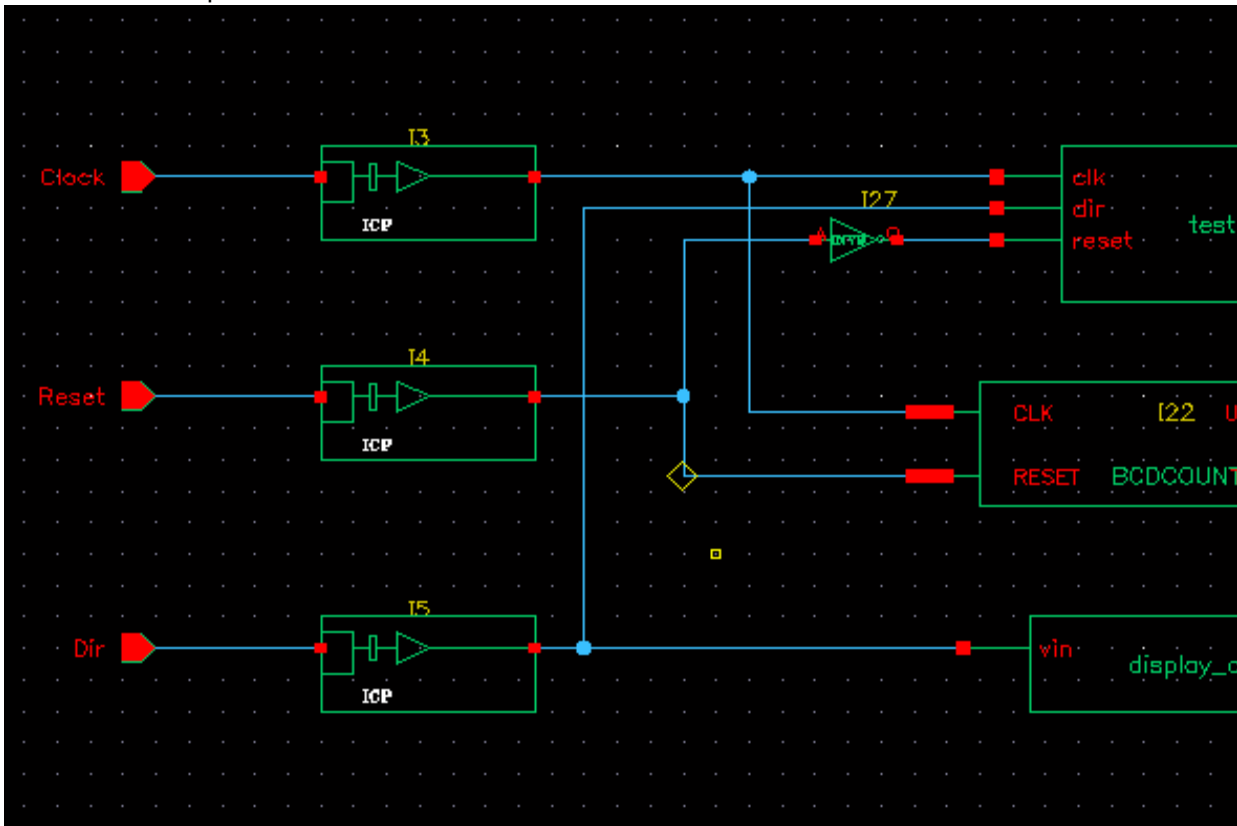


Fig: Final Input Pin Structure

7.4 Outputs

Place 14 instances of the IOLIB_4M BU2P on the right hand side of the schematic. We need to connect up all the output pins including the 4 bit buses from the BCDCOUNT.

7.4.1 Connecting Test1 outputs

Create 4 output pins labelled a b c d and place these to the right of the top BU2P elements.

Hint: Make sure you select “Outputs” for the pin direction!

Connect them to the adjacent BU2P elements.

Then wire the outputs of the test1 instance to corresponding BU2P inputs.

The final structure should resemble this:

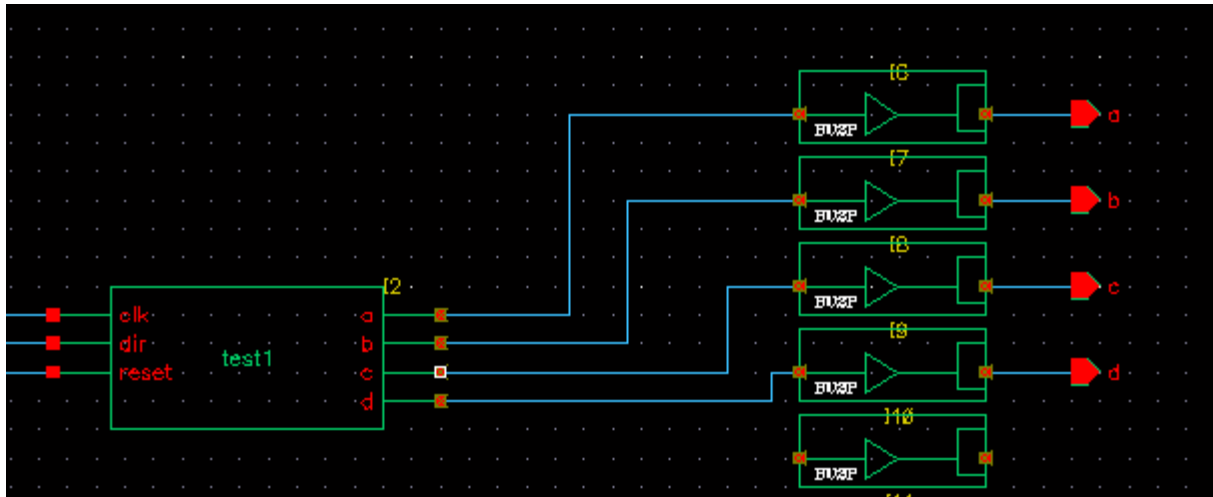


Fig: Schematic Structure test1 outputs

7.5 Connecting BCDCOUNT outputs

The BCDCOUNT has two 4 bit, <3:0> outputs and we need to label these as non bus elements for the outputs. We will call the outputs UNIT3. UNIT0 and TEN3... TEN0.

Create 8 output pins labelled UNIT3 UNIT2 UNIT1 UNIT0 TEN3 TEN2 TEN1 TEN0.

Hint: If you enter these names separated by spaces into the pin name you can place multiple pins. We could also use the bus expansion option to automatically generate individual pins from a bus definition i.e. UNIT<3:0>.

Wiring the buses from the BCDCOUNT to individual BU2P instances.

We need to split the bus to extract the individual wires. We do this by naming the bus elements using net labels.

First let us connect up all the elements. We do this by connecting a wire from the 4 bit output to all 4 of the relevant BU2P instances as shown below:

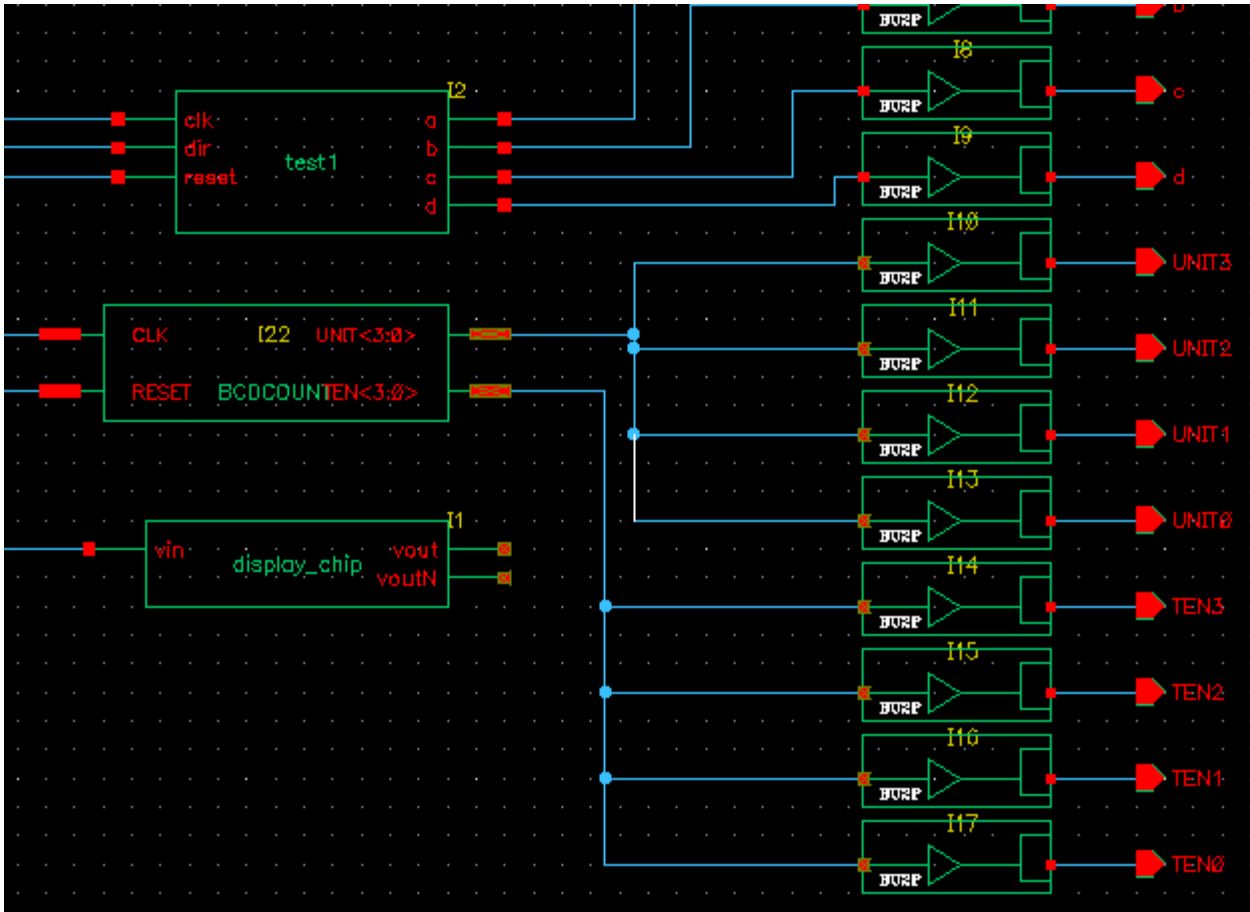


Fig: Schematic Unlabelled connections for the BCDCOUNT

7.5.1 Labelling the bus and single bit wires

In order to split the bus we need to use the Add -> Wire Name, menu or icon command.

7.5.1.1 Labelling the buses

We need to label the bus nets before we split them. We need to use the names UNIT<3:0> and TEN<3:0> for the respective nets. Use the Add -> Wire Name function and click on the section of the net just after the output pins on the BCDCOUNT instance. The result should resemble this:

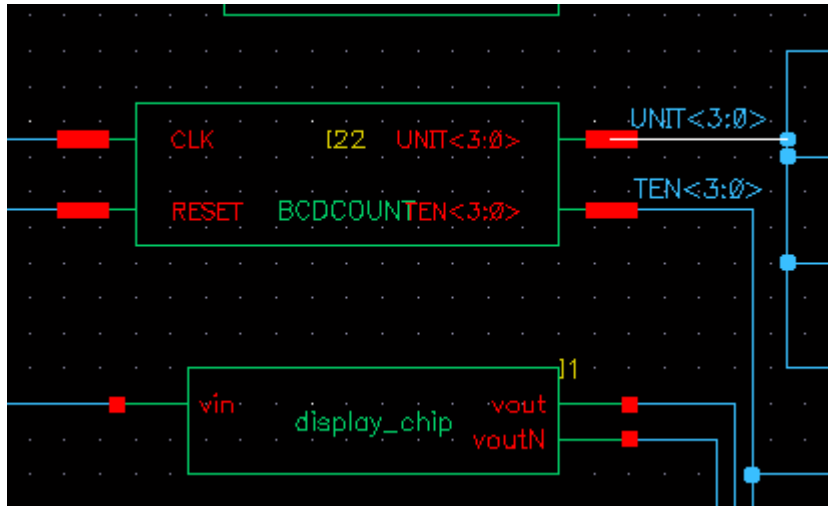


Fig: BCDCOUNT Bus Elements Labelled

7.5.1.2 UNIT

We need to add the following labels UNIT<3>, UNIT<2>, UNIT<1> and UNIT<0> on the correct wire connection.

7.5.1.3 TEN

We need to add the following labels TEN<3>, TEN<2>, TEN<1> and TEN<0> on the correct wire connection.

Hint: You can have multiple labels in the Wire Name form and add them automatically when you click on a wire.

You can also use the BUS Expansion mode to save typing. The first click will place the left hand net i.e. UNIT<3>

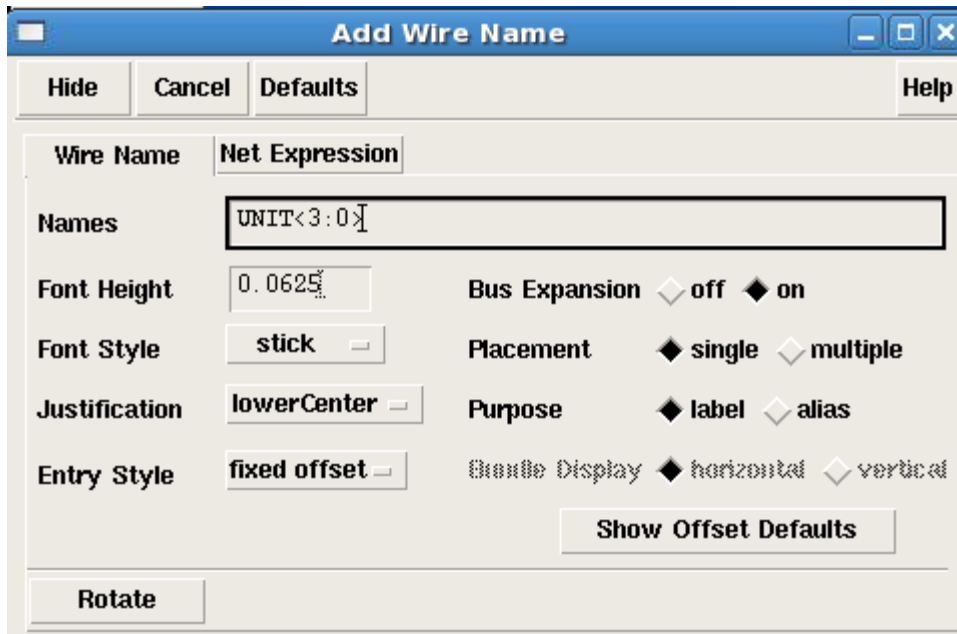


Fig: Wire Name with Bus Expansion enable for UNIT<3:0>

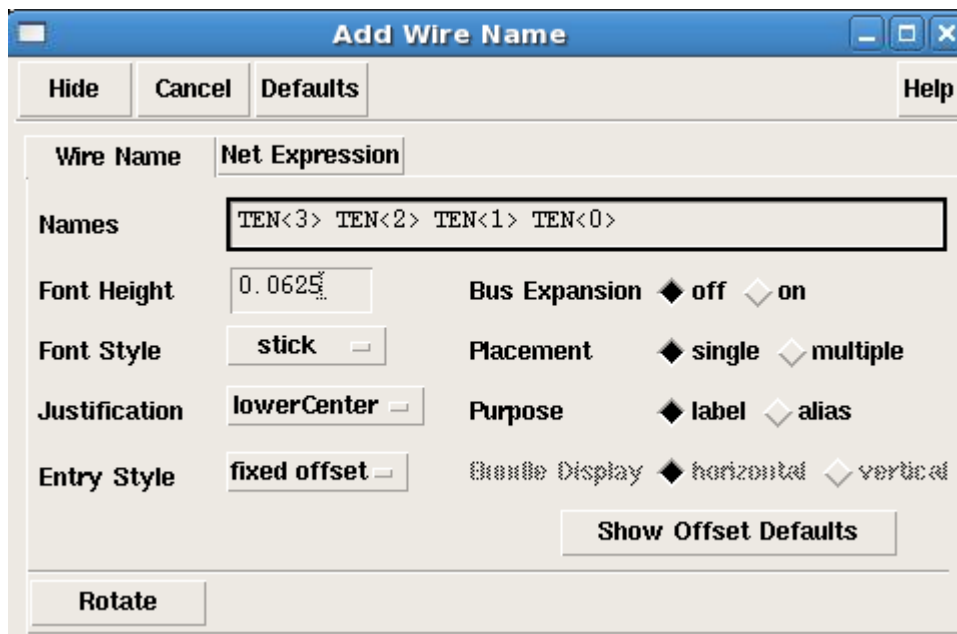


Fig: Wire Name without Bus Expansion enabled

The fully connected and labelled structure should resemble:

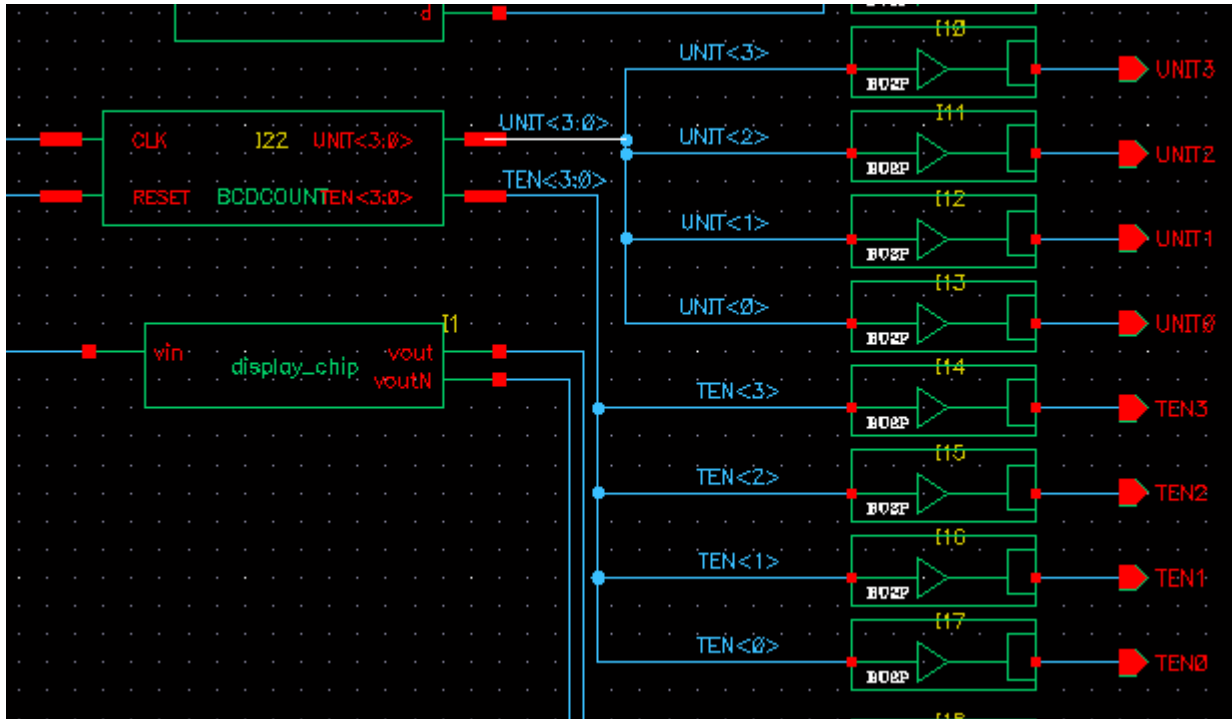


Fig: Connected and labelled BDCOUNT structure

7.6 Connecting display_chip Outputs

We need to place two output pins called vout and voutN. Connect these to the remaining BU2P outputs. Connect the outputs of the display_chip to the appropriate BU2P input.

7.7 Final Output Schematic Structure

The final output structure should resemble the one below:

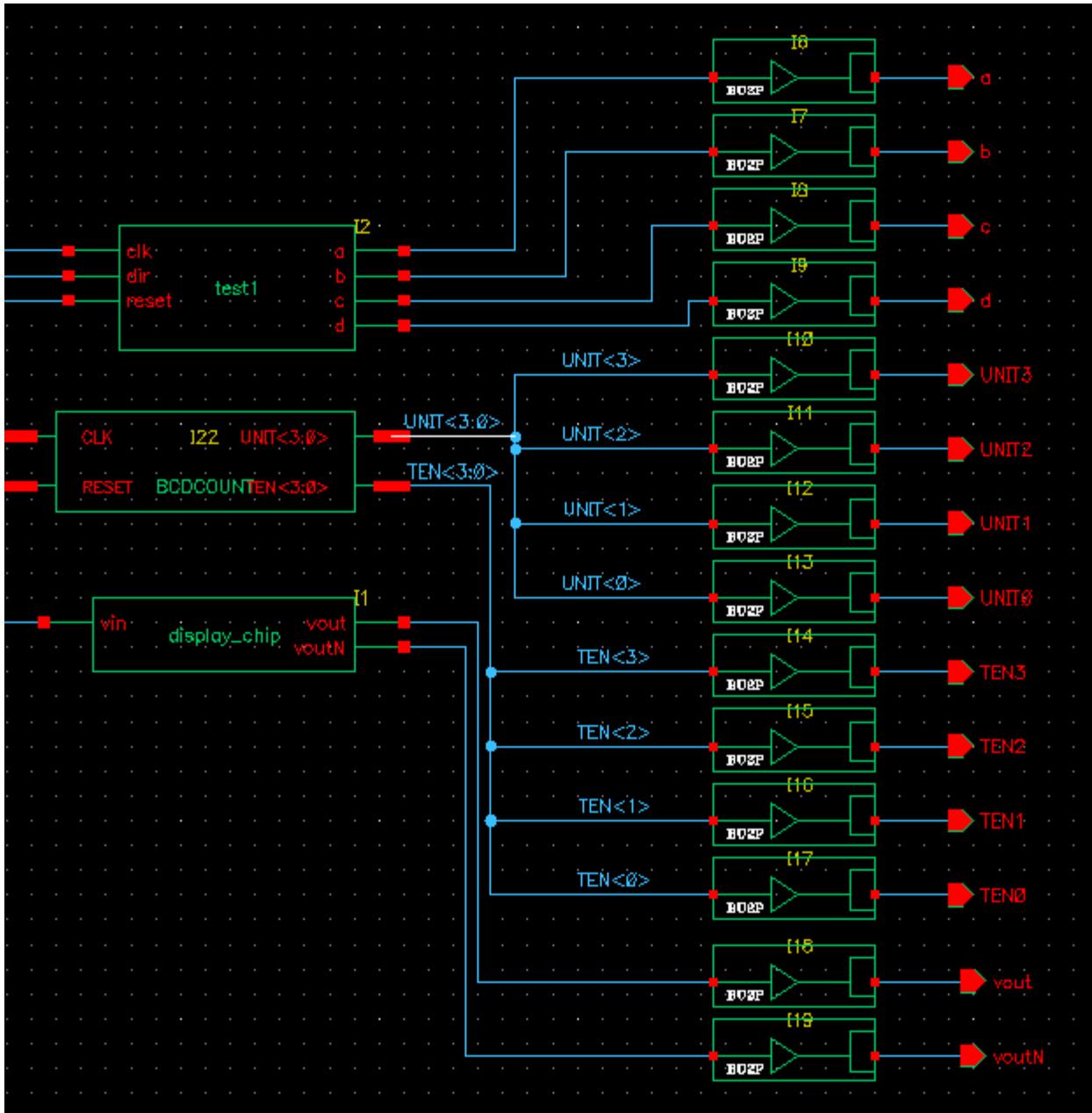


Fig: Final BCDCOUNT Schematic Structure

8 Adding Power

We need to add the power pads to the design. We are going to use the IOLIB_4M cells GND3ALLP and VDD3ALLP to connect all the standard power rails, both analogue and digital with a single cell. We need to define a power supply for these connections hence we will do the following:

- Place the IOLIB_4M GND3ALLP cell and connect it to an analogLib gnd symbol

- Place the IOLIB_4M VDD3ALLP cell and connect it to an analogLib vdd symbol

This section of the schematic should resemble that shown:

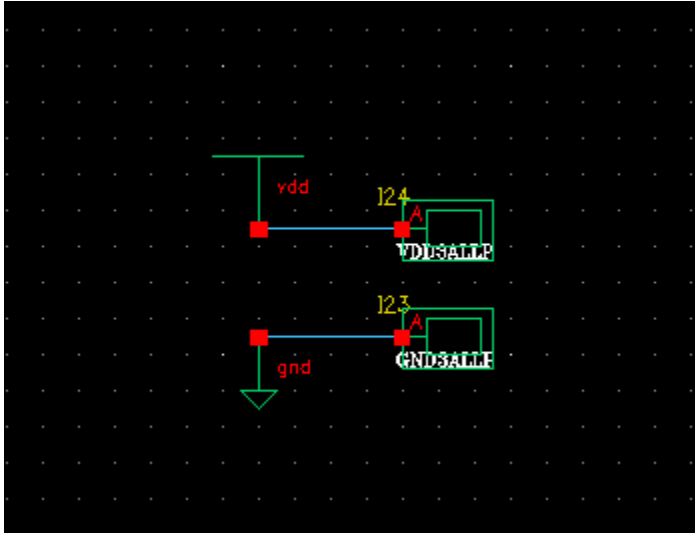


Fig: Power Pads Connected to Global Power Supplies

9 Final Schematic Structure

The final structure should be:

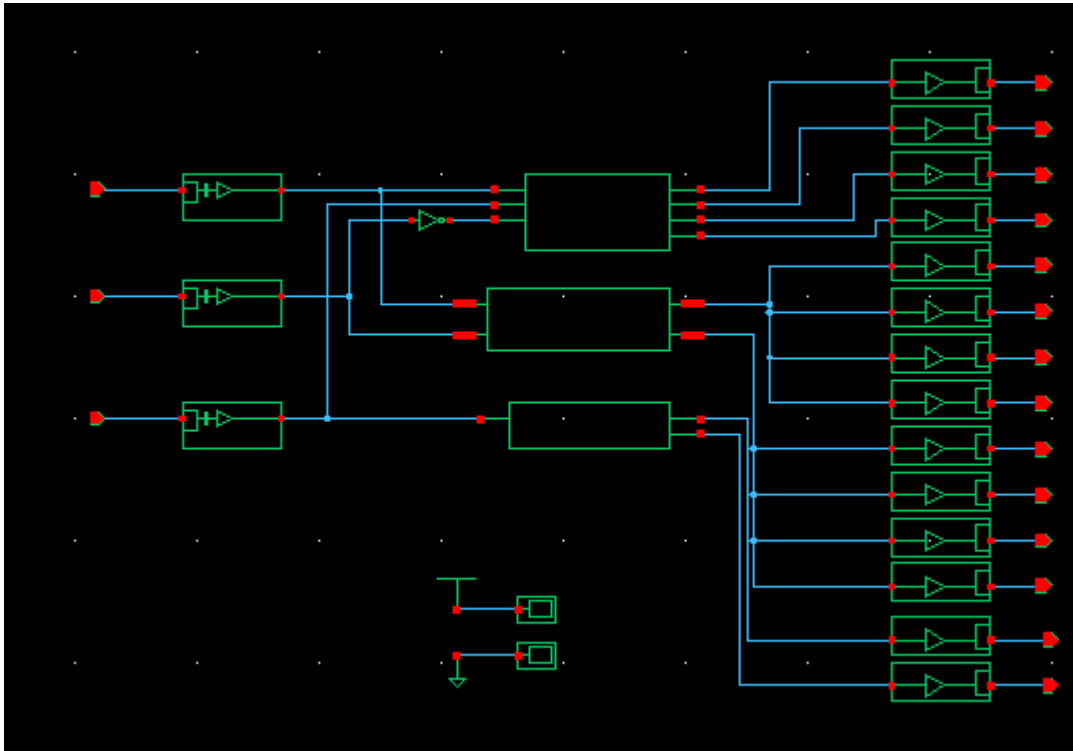


Fig: Final Schematic Structure

10 Digital Simulation

From the Schematic Window invoke the NCVerilog Simulator using the Tools -> Simulation -> NCVerilog command.

Using the Virtuoso Simulation for NCVerilog tool run the Command -> Initialize command.

Now use the Setup -> Netlist command on the Simulation tool to invoke the Setup Netlist form.

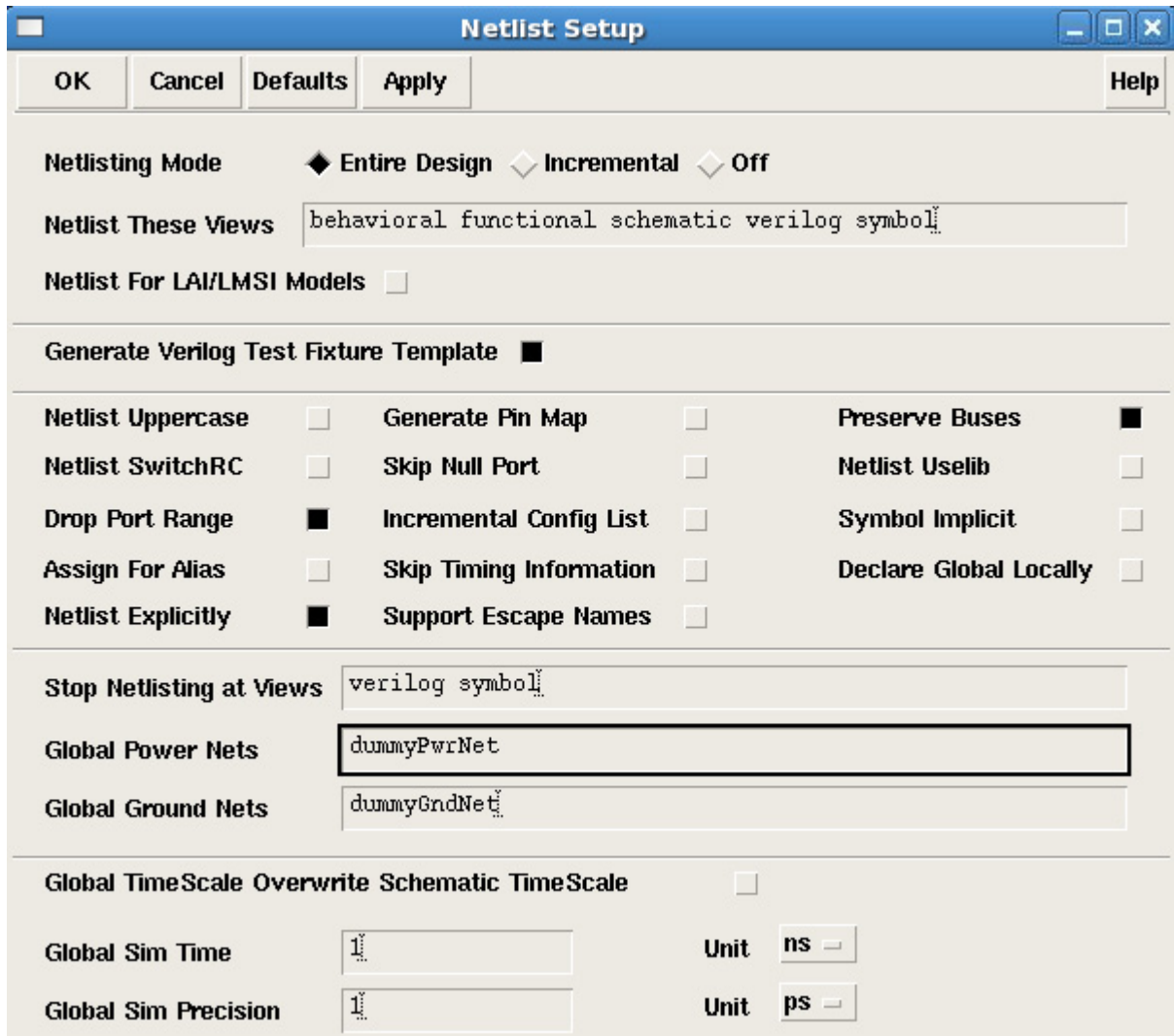


Fig: Setup Netlist form.

We need to add in the global power supplies the design kit uses. Add “gnd!” to the Global Ground Nets and “vdd!” to the Global Power Nets.

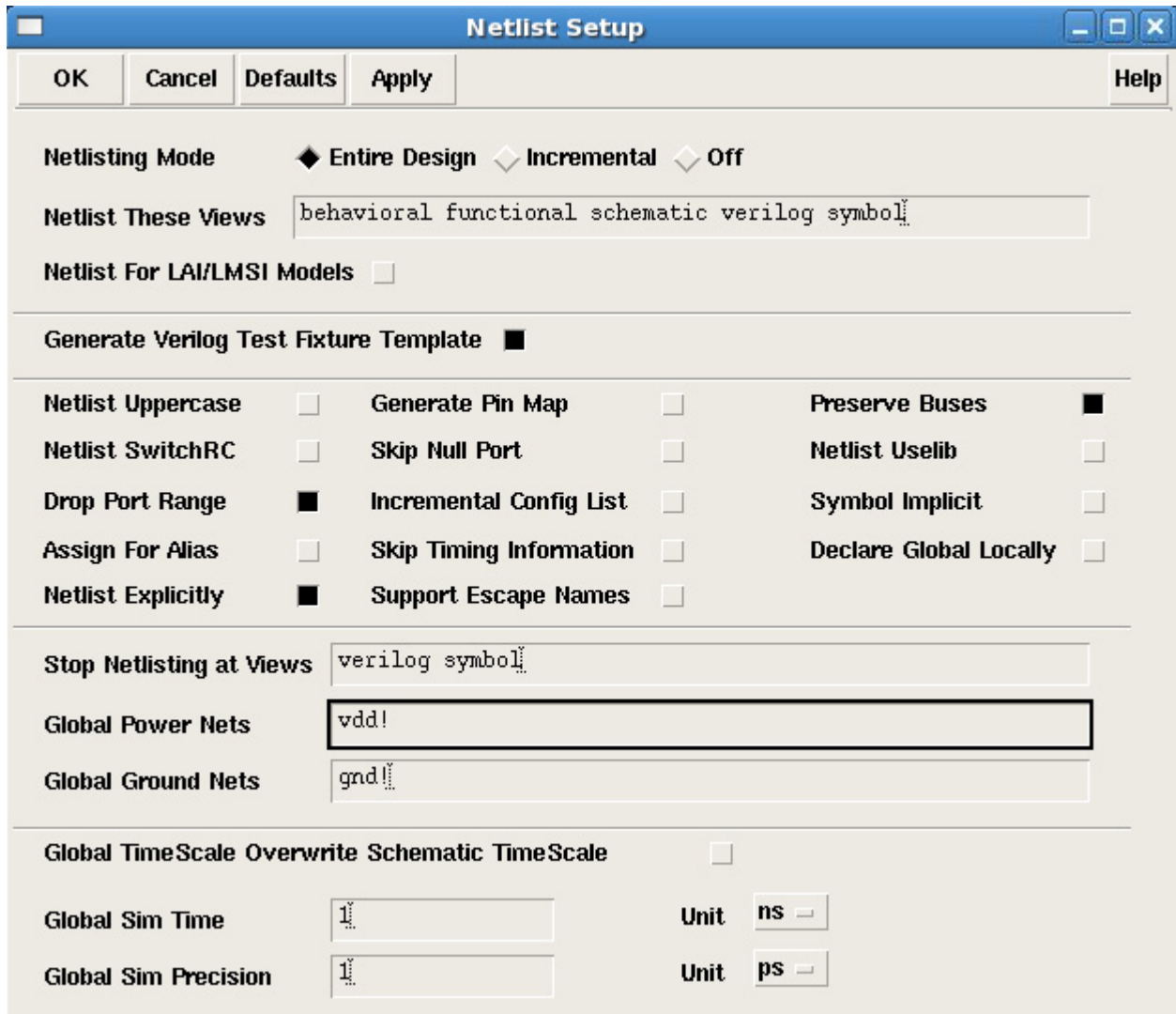


Fig: Netlist Setup Form with modified Global Power Nets

- OK the form and run the Commands -> Generate Netlist command.

10.1 Editing the Test Fixture

Use the Commands -> Edit Test Fixture to invoke the edit form. Leave the selection on the “Edit” and “Stimulus” and “OK” the form. To open up the Stimulus file for edit. We need to add in a reset, clock and direction sequence. The final stimulus file should resemble this.

- // Verilog stimulus file.
- // Please do not create a module in this file.
- // Default verilog stimulus.
-
- initial

Full Custom Using Standard Cells

```
• begin
•           Clock = 1'b0;
•           Dir = 1'b0;
•           Reset = 1'b0;
•           #10000 Reset = 1'b1;
•           #10000 Reset = 1'b0;
•           #1000000 Dir = 1'b1;
•           #1000000 $finish;
• end
•
• always
• begin
•           # 5000 Clock = ~ Clock;
• end
•
```

Fig: Stimulus File

Hint: To Paste into the Linux NEDIT window you need to use <CTRL> <SHIFT> v not <CTRL> v as you do with windows.

Run the simulation and use the Simvision Design Browser to select and display the inputs and outputs. Then run the Simulation-> Run, do a View -> Zoom -> Full X in the waveform viewer to see the entire waveform. The final result should resemble:

- Select Waveforms using Design Browser and Send to Waveform Browser
- Simulation -> Run
- View -> Zoom -> Full X

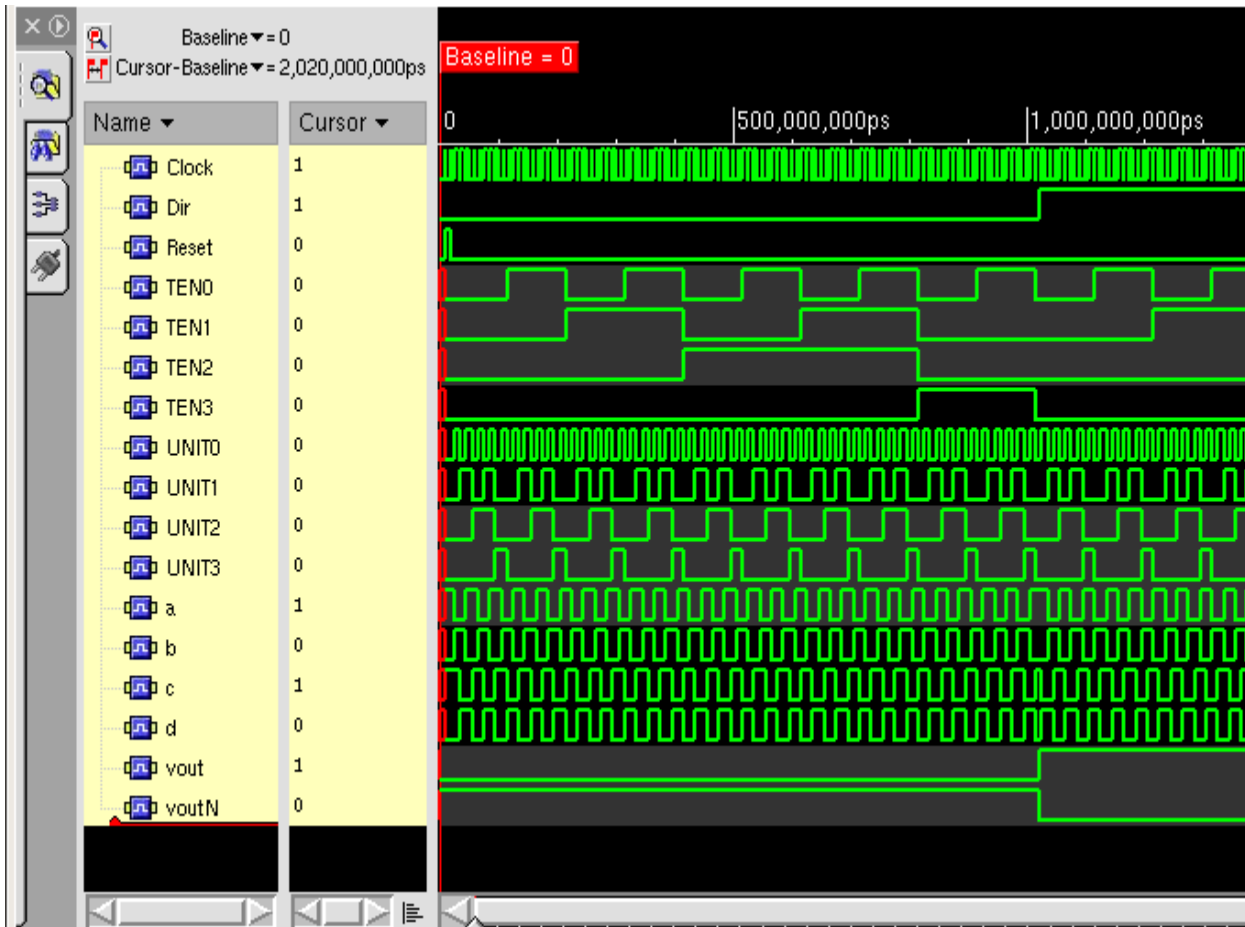


Fig: Output waveform from Digital Simulations

Hint: If it is not working make sure you put the inverter into the reset lines. Check the Reset State as well.

11 Generating a Full Chip Layout

11.1 Getting the required files.

The Encounter tool uses Verilog netlists and hence we need to extract the Verilog netlists from the simulation environment. The Virtuoso NCVerilog simulation tool has the directory in which the files are stored:

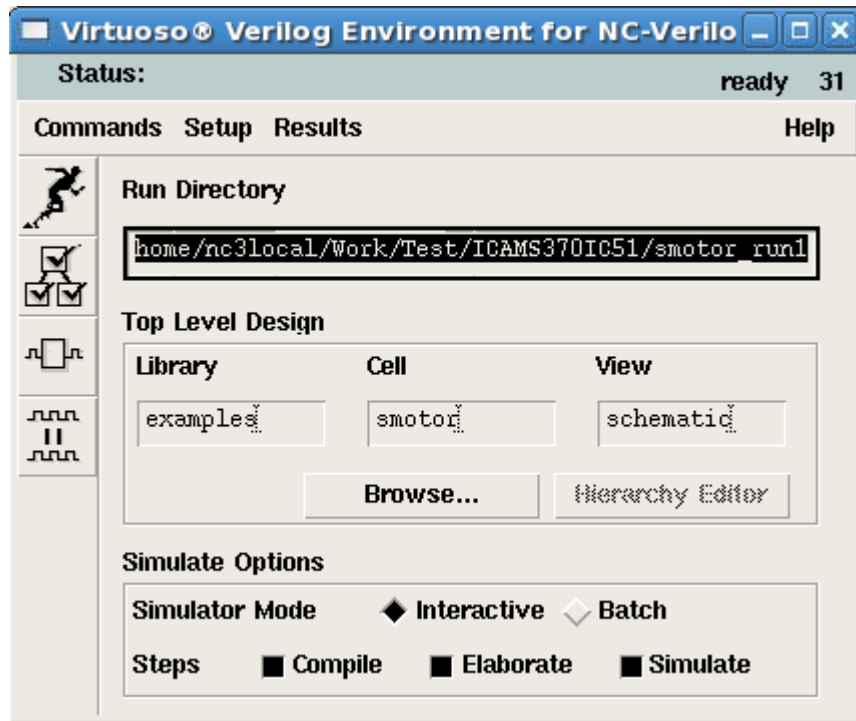


Fig: Showing file location for verilog files on Virtuoso Verilog Environment Tool

11.2 Extracting the files

cd into the directory indicated. Then cd into the underlying ihnl directory. You should see a display similar to this:

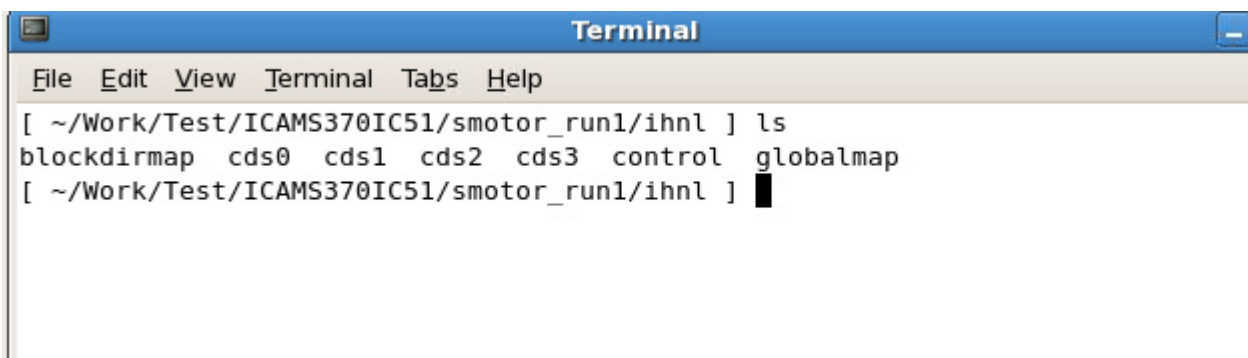


Fig: ihnl simulation directory ls command

Carefully run the following command, paying attention to the spaces

- `cat cds*/netlist >> ~/final.v`

- Spaces between cad and cds
- Spaces between netlist and >>
- Spaces between >> and ~
- No space between the "cds" and the "*"

The will copy and concatenate all the verilog netlist files into a single file called final.v in your home directory.

12 Configuring the Design Area : Now use the A4407Layout area

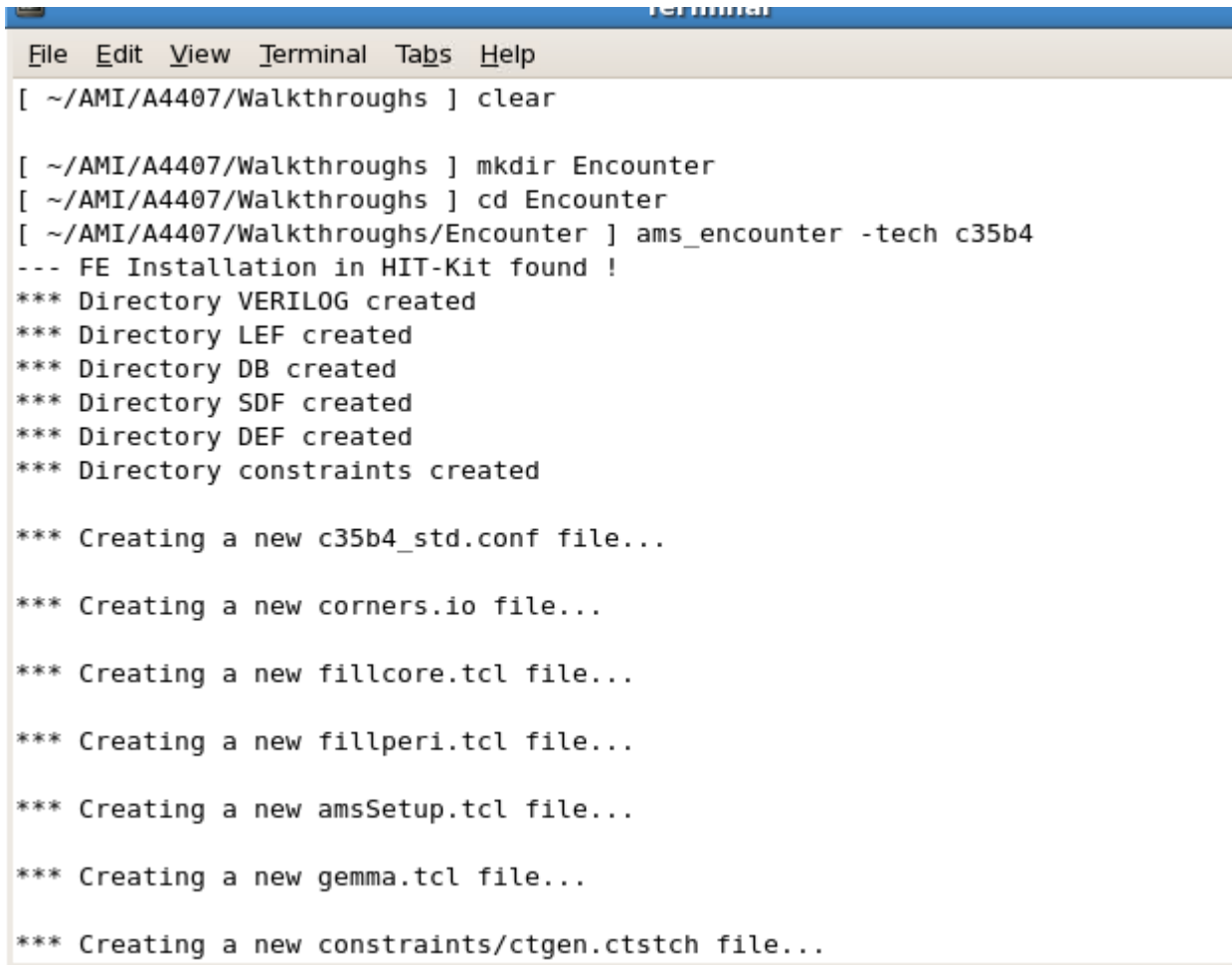
Use the transport tool to configure/open a terminal in the A4407Layout Walkthroughs area and then continue with the Walkthrough.

Return to your Walkthroughs area. Either by using cd or create a new window using the transport tool. Create a sub directory called "Encounter" using the mkdir command and move into it using the mv command.

- mkdir Encounter
- mv Encounter

Move into this directory and use the AMS configuration script ams_encounter -tech c35b4 to create the correct structures and environment for the encounter tool.

- ams_encounter -tech c35b4



```
File Edit View Terminal Tabs Help
[ ~/AMI/A4407/Walkthroughs ] clear

[ ~/AMI/A4407/Walkthroughs ] mkdir Encounter
[ ~/AMI/A4407/Walkthroughs ] cd Encounter
[ ~/AMI/A4407/Walkthroughs/Encounter ] ams_encounter -tech c35b4
--- FE Installation in HIT-Kit found !
*** Directory VERILOG created
*** Directory LEF created
*** Directory DB created
*** Directory SDF created
*** Directory DEF created
*** Directory constraints created

*** Creating a new c35b4_std.conf file...

*** Creating a new corners.io file...

*** Creating a new fillcore.tcl file...

*** Creating a new fillperi.tcl file...

*** Creating a new amsSetup.tcl file...

*** Creating a new gemma.tcl file...

*** Creating a new constraints/ctgen.ctstch file...
```

Fig: Encounter area preparation

12.1 Transferring the Verilog File

We need to copy the top level verilog file from our home directory into the VERILOG directory.

Use the command

- `cp ~/final.v VERILOG`

This will copy the file to the VERILOG sub directory, which is automatically created in the Encounter directory by the `ams_encounter` command.

12.2 Fixing and Editing the Verilog File global nets

When created Cadence Verilog used its own approach to implementing global nets in Verilog. Which cause encounter some significant problems and need to be fixed before we read in the design.

Use `nedit` to open the `final.v` file the command from the Encounter directory would be:

Full Custom Using Standard Cells

- nedit VERILOG/final.v
- Use the Search -> Replace option to invoke the Replace/Find form.
- We need to replace the following items in the entire file or “Window”.
 - cds_globals.vdd_ with vdd
 - cds_globals.gnd_ with gnd

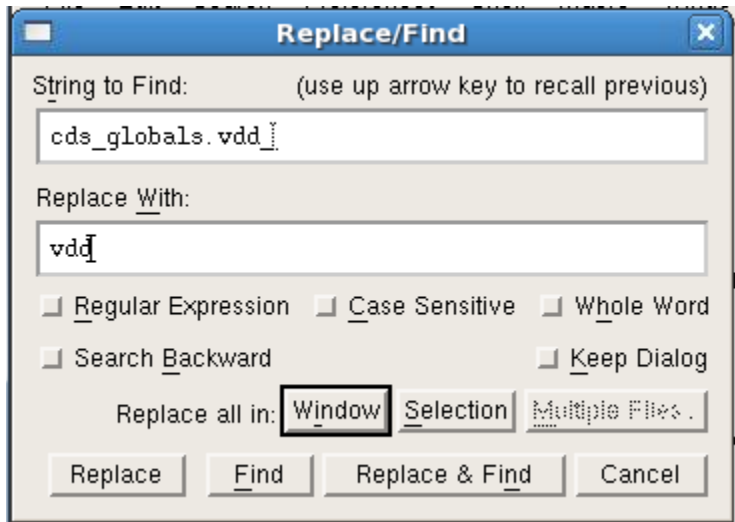


Fig: Nedit Replace/Find form configured to replace cds_globals.vdd with vdd

If you are unsure of the find them in the netlist. Check that the replace has worked correctly for all instances.

12.2.1 Starting Encounter

To invoke encounter use the following command:

```
encounter
```

This will start the graphic tool. In addition the prompt on the terminal in which you typed the command will change to indicate it is now the encounter tools input/output stream.

Full Custom Using Standard Cells

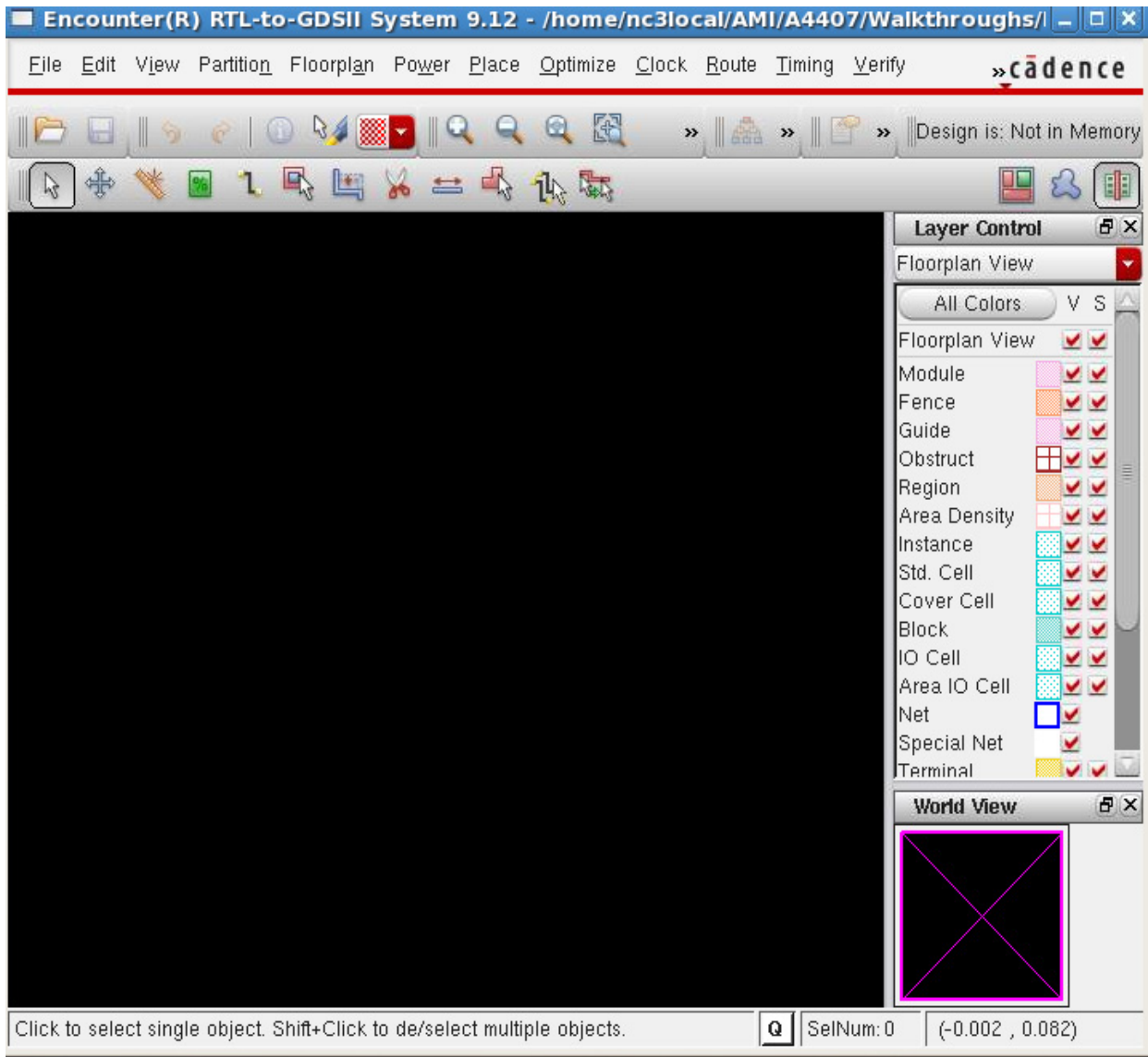
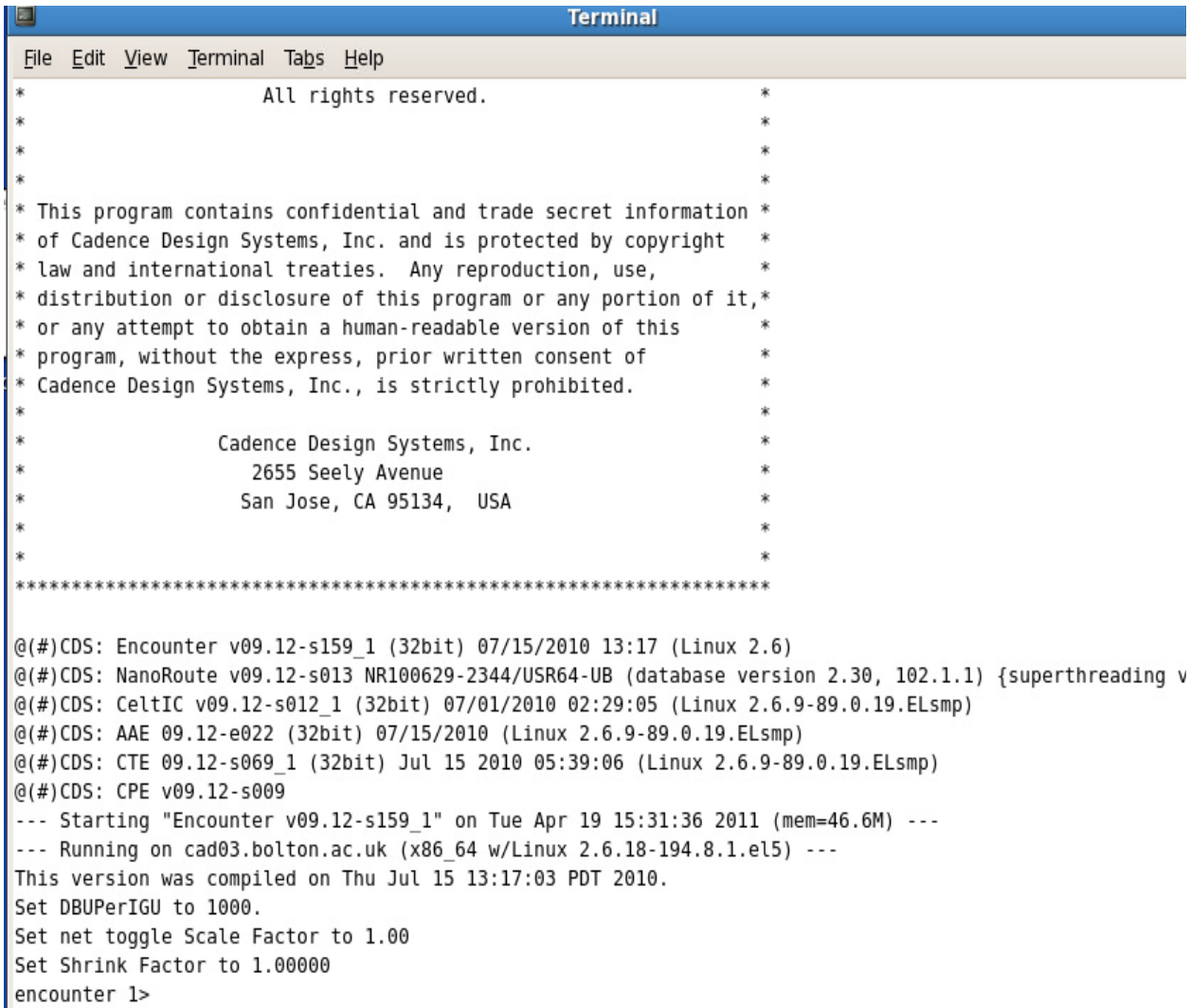


Fig: Encounter graphics tool



```
Terminal
File Edit View Terminal Tabs Help
*           All rights reserved.           *
*                                           *
*                                           *
*                                           *
* This program contains confidential and trade secret information *
* of Cadence Design Systems, Inc. and is protected by copyright *
* law and international treaties. Any reproduction, use, *
* distribution or disclosure of this program or any portion of it,*
* or any attempt to obtain a human-readable version of this *
* program, without the express, prior written consent of *
* Cadence Design Systems, Inc., is strictly prohibited. *
*                                           *
*           Cadence Design Systems, Inc. *
*           2655 Seely Avenue *
*           San Jose, CA 95134, USA *
*                                           *
*****
@(#)CDS: Encounter v09.12-s159_1 (32bit) 07/15/2010 13:17 (Linux 2.6)
@(#)CDS: NanoRoute v09.12-s013 NR100629-2344/USR64-UB (database version 2.30, 102.1.1) {superthreading v
@(#)CDS: CeltIC v09.12-s012_1 (32bit) 07/01/2010 02:29:05 (Linux 2.6.9-89.0.19.ELsmp)
@(#)CDS: AAE 09.12-e022 (32bit) 07/15/2010 (Linux 2.6.9-89.0.19.ELsmp)
@(#)CDS: CTE 09.12-s069_1 (32bit) Jul 15 2010 05:39:06 (Linux 2.6.9-89.0.19.ELsmp)
@(#)CDS: CPE v09.12-s009
--- Starting "Encounter v09.12-s159_1" on Tue Apr 19 15:31:36 2011 (mem=46.6M) ---
--- Running on cad03.bolton.ac.uk (x86_64 w/Linux 2.6.18-194.8.1.el5) ---
This version was compiled on Thu Jul 15 13:17:03 PDT 2010.
Set DBUPerIGU to 1000.
Set net toggle Scale Factor to 1.00
Set Shrink Factor to 1.00000
encounter 1>
```

Fig: Encounter Terminal Window initialisation

12.3 Basic Configuration

Use the Encounter graphics tool command File -> Import design to open the design import tool.

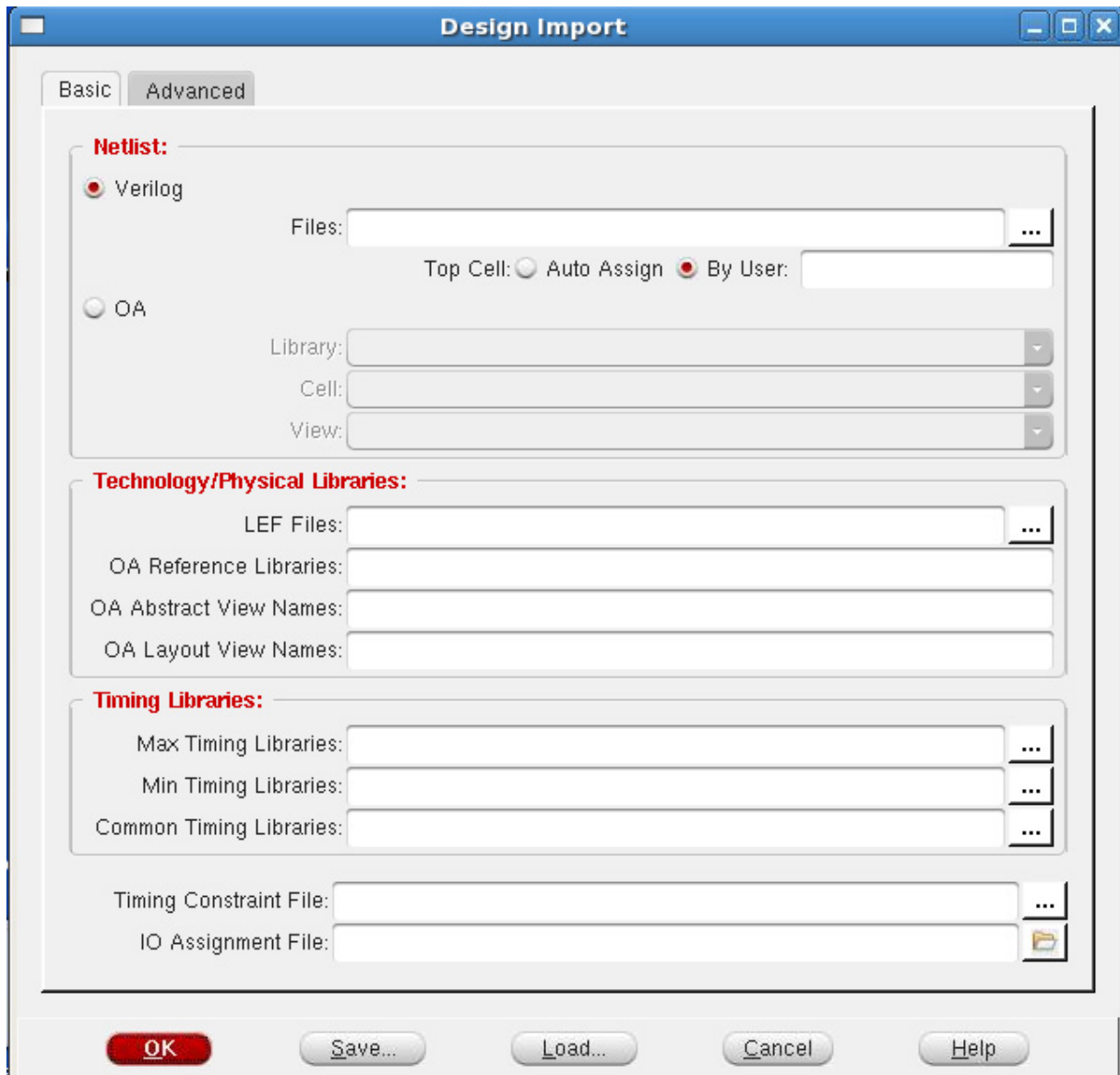


Fig: Design Import Encounter form

Select the “Load” option at the base of this form to bring up the Load Import Configuration form.

Select the c35b4_std.conf and then “Open” the Load Import Configuration form to load the configuration and return to the Design Import Form.

Full Custom Using Standard Cells

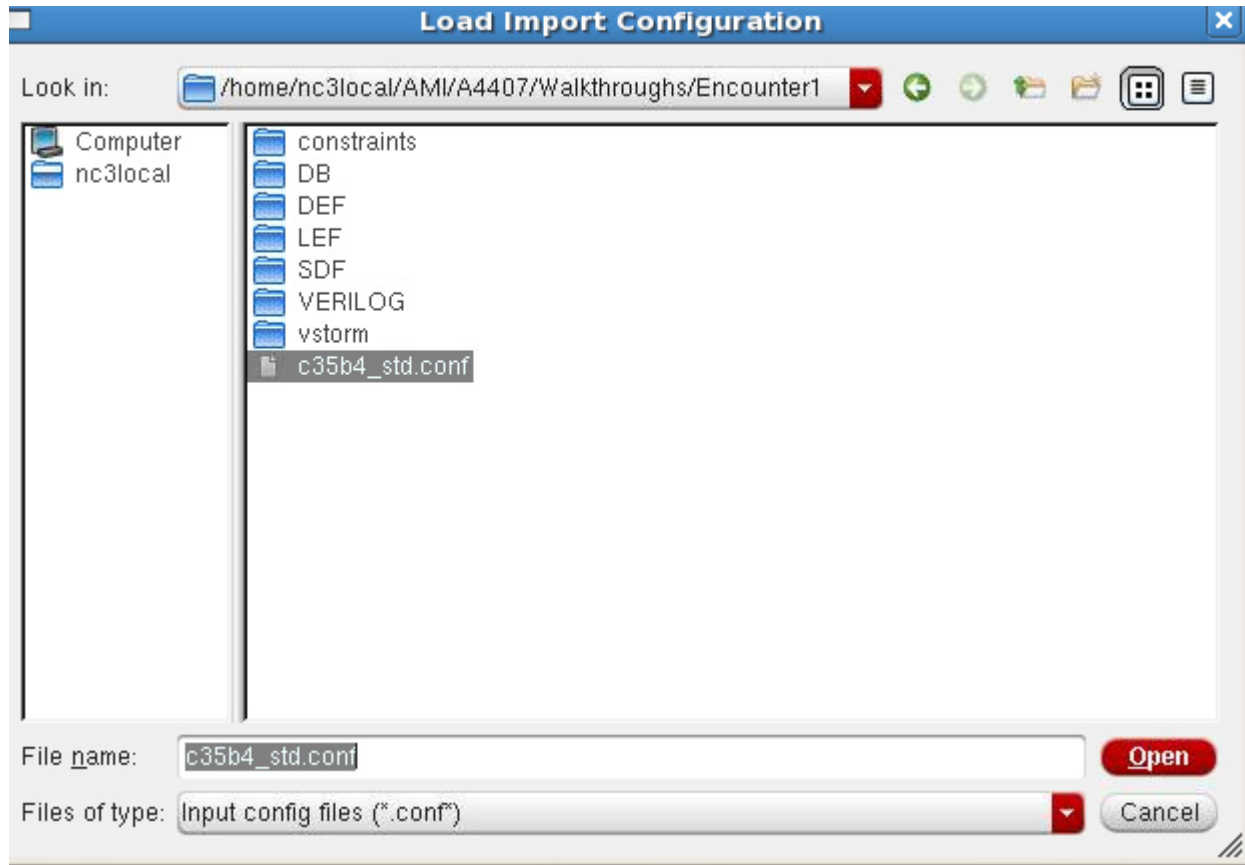


Fig: Load Import Configuration form with c35b4_std.conf selected

The Import Design form should now look like this:

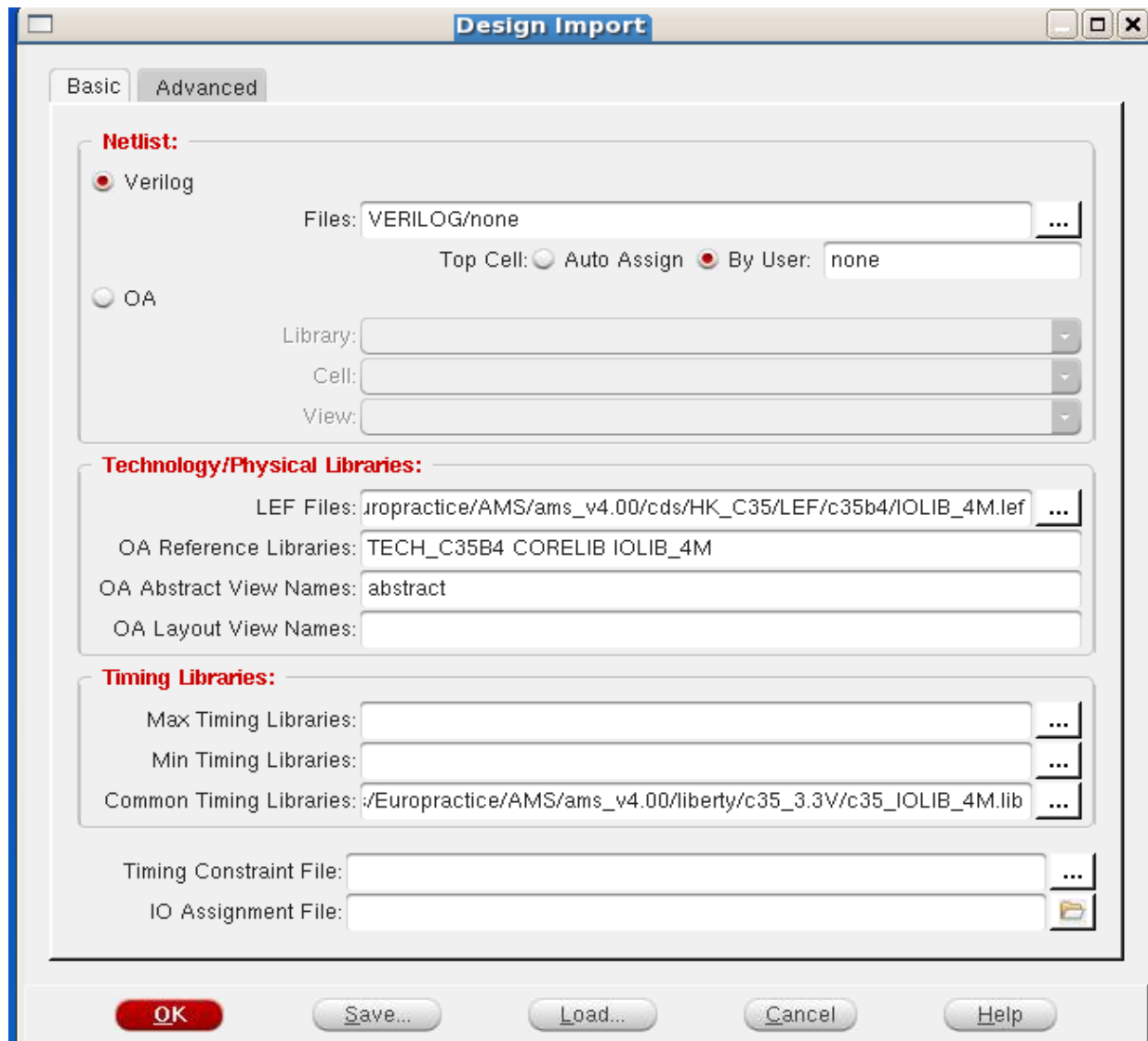


Fig: Basic Configuration on the Design Import Form

12.3.1 Make the following changes to the form

In the Netlist section at the top

- **Change the Files to Verilog/final.v**

Either by typing or using the “...” icon on the Right hand side and opening the browser

- **Change the Top Cell Entry to Top Cell**

Be aware the use methodology for the browser is unusual. Requiring you to "Add" the new file and "Delete" the "Verilog/none" entry.

12.3.1.1 Advanced Options

Now select the Advanced tab at the top and then select the “Power” item in the left hand pane.

- Add “gnd” to the ground nets and “vdd” to the power nets.

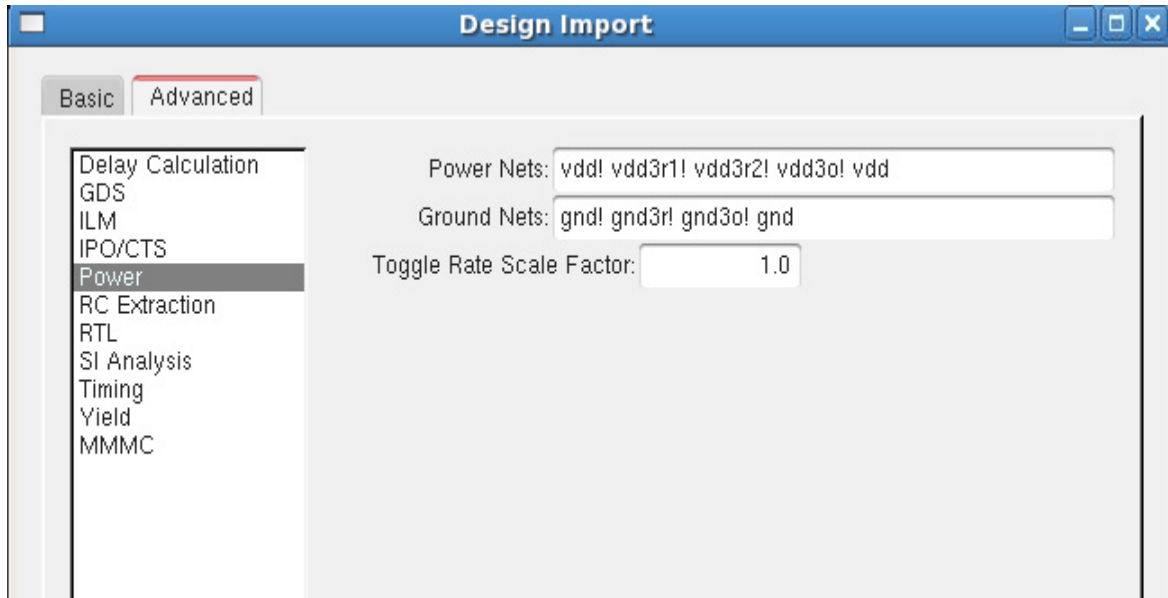


Fig: Design Import Advanced Tab with Power nets selected and gnd and vdd added

12.3.1.2 Saving our work on the configuration file

Now select the save option and save the changes to a new “.conf” file. So if we have to repeat this stage we can load this instead of the c35b4_std.conf form. Try and remember the name.

12.4 Implementing the changes

Now "OK" the form to action the settings in the Design Import Form. This should import the Verilog file and create a basic layout of the chip. At this point the cells are not placed and the IO pad placement is default.

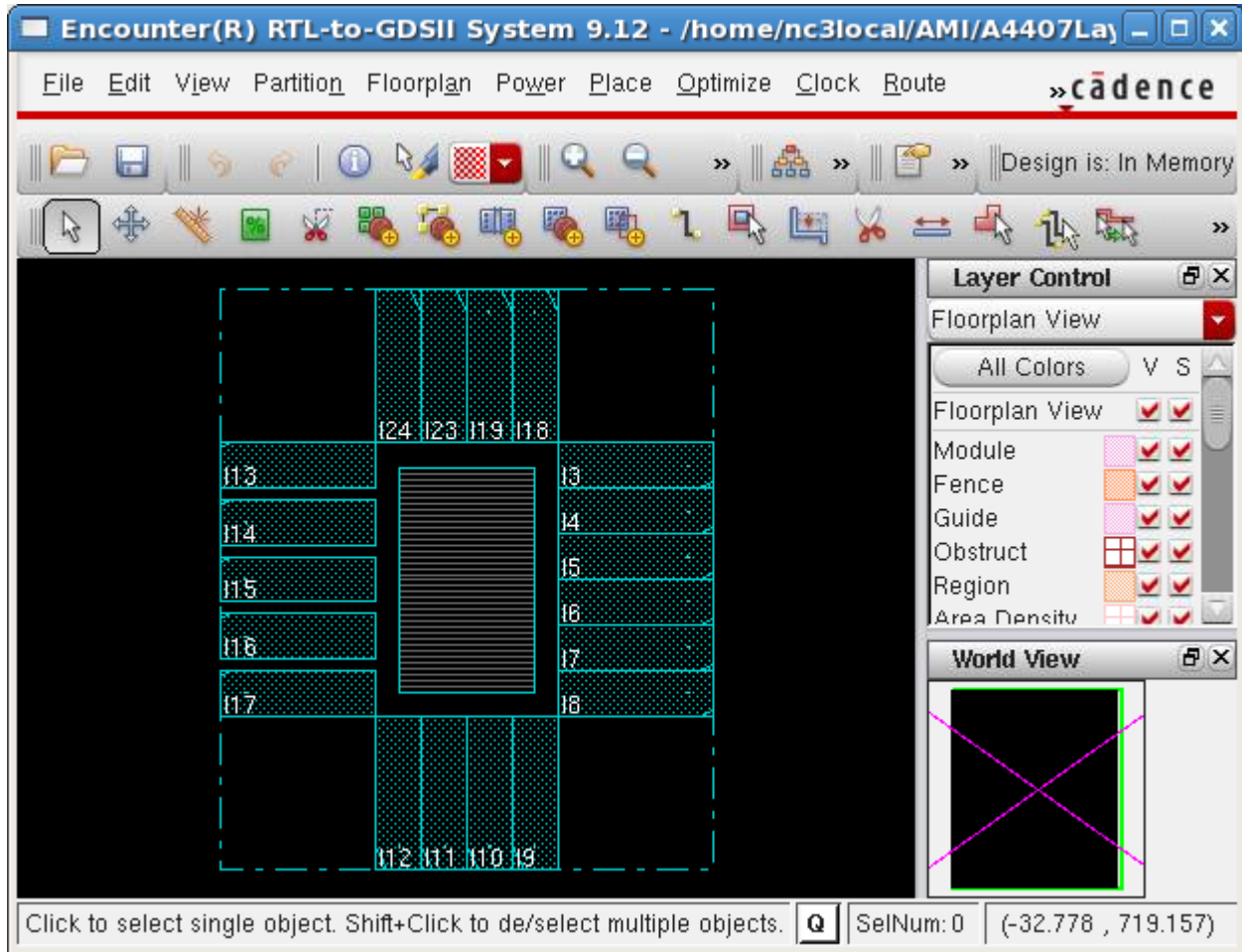


Fig: Encounter after Configured File Import Function

12.5 Generating and Using the IO file to control the location of peripheral cells.

12.5.1 Putting the Power Pads where we want them

In this design the location of the inputs and outputs is not really an issue. However we do want to ensure that we locate the Power Pads in fixed location. Ideally on the top/bottom or sides of the chip. To do this we first need to create the default IO pads file. This saves us having to create our own from scratch. As IO pads have a common size and spacing, in normal standard cell design we can "swap" pads relatively easily. We could use the graphic interface to do this however in this example we will use the script file.

12.5.2 Creating the IO Pads file

Use the File -> Save -> IO option to open up the save form and use this to save the io file.

- The default name will be: smotor.save.io

Full Custom Using Standard Cells

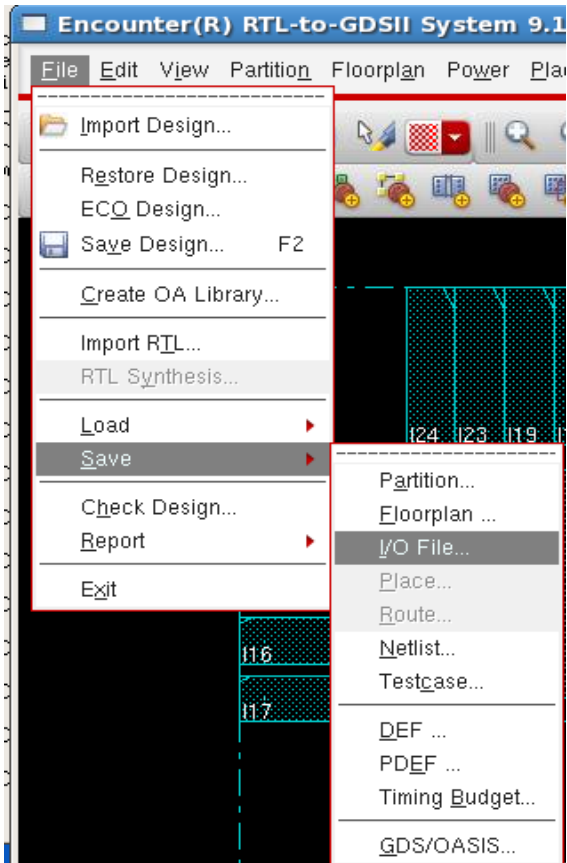


Fig: File Save IO File Encounter Options

12.5.3 Setting up the IO pads

Return to the encounter terminal window and use nedit to open the VERILOG/final.v file. Use the command:

- `nedit VERILOG/final.v &`

Hint: The & means run the command in background and lets you do other things in the window at the same time.

Look at the file and make a note of the instance names for the VDD and Ground pins. In my case they were:

- `VDD3ALLP I24 (.A(vdd));`
- `GND3ALLP I23 (.A(gnd));`

12.5.4 Editing the smotor.io.save file

Now close the final.v file and using nedit again open the smotor.io.save file. It should look something like this:

Full Custom Using Standard Cells

```
• #####
• # Generated by: Cadence Encounter 09.12-s159_1
• # OS: Linux x86_64(Host ID cad03.bolton.ac.uk)
• # Generated on: Tue Apr 19 16:37:31 2011
• # Command: saveloFile -locations smotor.save.io
• #####
•
• (globals
•   version = 3
•   io_order = default
• )
• (iopad
•   (top
•     (inst name="I24"offset=340.4000 place_status=placed )
•     (inst name="I23"offset=440.4000 place_status=placed )
•     (inst name="I19"offset=540.4000 place_status=placed )
•     (inst name="I18"offset=640.4000 place_status=placed )
•   )
•   (left
•     (inst name="I17"offset=340.4000 place_status=placed )
•     (inst name="I16"offset=465.4000 place_status=placed )
•     (inst name="I15"offset=590.4000 place_status=placed )
•     (inst name="I14"offset=715.4000 place_status=placed )
•     (inst name="I13"offset=840.4000 place_status=placed )
•   )
•   (bottom
•     (inst name="I12"offset=340.4000 place_status=placed )
•     (inst name="I11"offset=440.4000 place_status=placed )
•     (inst name="I10"offset=540.4000 place_status=placed )
•     (inst name="I9"offset=640.4000 place_status=placed )
•   )
•   (right
•     (inst name="I8"offset=340.4000 place_status=placed )
•     (inst name="I7"offset=440.4000 place_status=placed )
•     (inst name="I6"offset=540.4000 place_status=placed )
•     (inst name="I5"offset=640.4000 place_status=placed )
•     (inst name="I4"offset=740.4000 place_status=placed )
```

Full Custom Using Standard Cells

- (inst name="I3" offset=840.4000 place_status=placed)
-)
-)

We want the VDD and GND on opposite sides of the core. So in this example we need to move either I24 or I23 and swap it with one of the elements on the bottom. The joy of standard size elements is we can do this easily. We could do this manually/graphically using the Encounter tools and then update the IO file. However in this case we will use a text based approach.

So swap the names I11 and I23

It would be nice if VDD I24 was more central as well so swap I10 and I24.

So Swap the names I19 and I24.

This will give us a final version of:

- (iopad
- (top
- (inst name="I19"offset=340.4000 place_status=placed)
- (inst name="I11"offset=440.4000 place_status=placed)
- (inst name="I24"offset=540.4000 place_status=placed)
- (inst name="I18"offset=640.4000 place_status=placed)
-)
- (left
- (inst name="I17"offset=340.4000 place_status=placed)
- (inst name="I16"offset=465.4000 place_status=placed)
- (inst name="I15"offset=590.4000 place_status=placed)
- (inst name="I14"offset=715.4000 place_status=placed)
- (inst name="I13"offset=840.4000 place_status=placed)
-)
- (bottom
- (inst name="I12"offset=340.4000 place_status=placed)
- (inst name="I23"offset=440.4000 place_status=placed)
- (inst name="I10"offset=540.4000 place_status=placed)
- (inst name="I9" offset=640.4000 place_status=placed)
-)
- (right
- (inst name="I8" offset=340.4000 place_status=placed)
- (inst name="I7" offset=440.4000 place_status=placed)

- (inst name="I6" offset=540.4000 place_status=placed)
- (inst name="I5" offset=640.4000 place_status=placed)
- (inst name="I4" offset=740.4000 place_status=placed)
- (inst name="I3" offset=840.4000 place_status=placed)
-)
-)

12.5.5 Save the changes and Exit the editor

Save this configurations as my_smotor.conf

- **Save this as my_smotor.conf**

Now close the editor.

12.6 Including the changes in your Design Import configuration file

Now return to, or reopen the Design Import form. Using the Menu File -> Import command. At the base of the form is the IO Assignment entry. Use the file browser on the right hand side to select the my_smotor.conf file. Which should now exist in your area.

Use the form "Save" option to save a new configuration file, called to my_smotorv2.conf.

12.7 Using the new configuration file to create the design

Now exit encounter and run the following command to re-invoke encounter using your updated configuration file to create the design with all the changes we have defined.

- **encounter -config my_smotorv2.conf**

It should run with no errors. If we now look at the placement of the instances in the graphic tool we will see that VDD and Ground, I24 and I23 are on opposite sides of the chip.

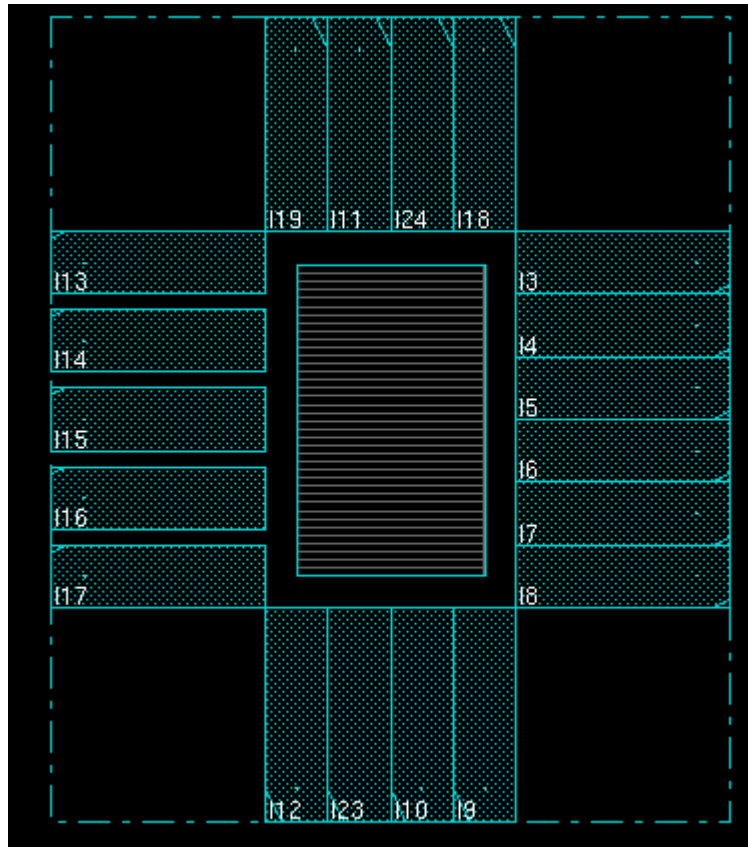


Fig: Showing changed I23 and I24 placements

12.8 Configuring the AMS scripts

Before we use the AMS scripts we need to add in our power supply settings to prevent them being overwritten. We could manually fix the result but this avoids potential issues if we rerun the procedure.

In the Encounter terminal window use the nedit command to edit the amsSetup.tcl file.

```
nedit amsSetup.tcl
```

Look for the amsGlobalConnect procedure. We need to add the following entries into the core and both sections.

- `globalNetConnect vdd -type pggpin -pin vdd! -inst * -module {}`
- `globalNetConnect gnd -type pggpin -pin gnd! -inst * -module {}`

The final result should resemble:

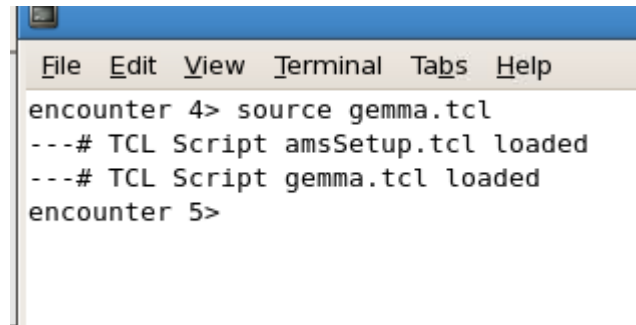
- `proc amsGlobalConnect type {`
- `##--- Define global power connects`
- `switch $type {`

```
• "core" {
•     ##--- Define global Power nets - make global connections
•     clearGlobalNets
•     globalNetConnect vdd! -type pgin -pin vdd! -inst * -module {}
•     globalNetConnect gnd! -type pgin -pin gnd! -inst * -module {}
•     globalNetConnect vdd -type pgin -pin vdd! -inst * -module {}
•     globalNetConnect gnd -type pgin -pin gnd! -inst * -module {}
•     }
• "both" {
•     ##--- Define global Power nets - make global connections
•     clearGlobalNets
•     globalNetConnect vdd! -type pgin -pin vdd! -inst * -module {}
•     globalNetConnect gnd! -type pgin -pin gnd! -inst * -module {}
•     globalNetConnect vdd3o! -type pgin -pin vdd3o! -inst * -module {}
•     globalNetConnect vdd3r1! -type pgin -pin vdd3r1! -inst * -module {}
•     globalNetConnect vdd3r2! -type pgin -pin vdd3r2! -inst * -module {}
•     globalNetConnect gnd3o! -type pgin -pin gnd3o! -inst * -module {}
•     globalNetConnect gnd3r! -type pgin -pin gnd3r! -inst * -module {}
•     globalNetConnect vdd -type pgin -pin vdd! -inst * -module {}
•     globalNetConnect gnd -type pgin -pin gnd! -inst * -module {}
•     }
• }
• }
```

12.8.1 Loading the AMS scripts

To load the scripts use the following command in the encounter terminal windows

- source gemma.tcl

A screenshot of a terminal window titled 'Encounter'. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', 'Tabs', and 'Help'. The terminal text shows the command 'source gemma.tcl' being executed, followed by two status messages: '--# TCL Script amsSetup.tcl loaded' and '--# TCL Script gemma.tcl loaded'. The prompt 'encounter 5>' is visible at the bottom.

```
encounter 4> source gemma.tcl
---# TCL Script amsSetup.tcl loaded
---# TCL Script gemma.tcl loaded
encounter 5>
```

Fig: Encounter terminal response to gemma.tcl

If successful and you type the command `ha` in the encounter window you will see the basic guidance. Which tells you what each stage actually does.

- --# amc start end
- --# 1 - amsDbSetup
- --# 2 - amsUserGrid
- --# 3 - amsGlobalConnect both
- --# 4 - amsOpCond minmax
- --# 5 - amsLoadCons func
- --# 6 - amsFloorplan peri 0.8 50
- --# 7 - amsPowerRoute {{vdd! 20} {gnd! 20}}
- --# 8 - amsAddEndCaps
- --# 9 - amsPlace ntd
- --# 10 - amsCts
- --# 11 - amsTa postCTS {func test}
- --# 12 - amsOpt postCTS hold {func test}
- --# amsOpt postCTS drv {func test}
- --# amsTa postCTS {func test}
- --# 13 - amsFillcore
- --# 14 - amsFillperi
- --# 15 - amsRoute wroute
- --# 16 - amsTa postRoute {func test}
- --# 17 - amsWrite final
- --# 18 - amsWriteSDF
- #####

Type the following command to run them all at once or run them one at a time.

- amc 3 6

Commands 1 and 2 have been run by the configuration script.

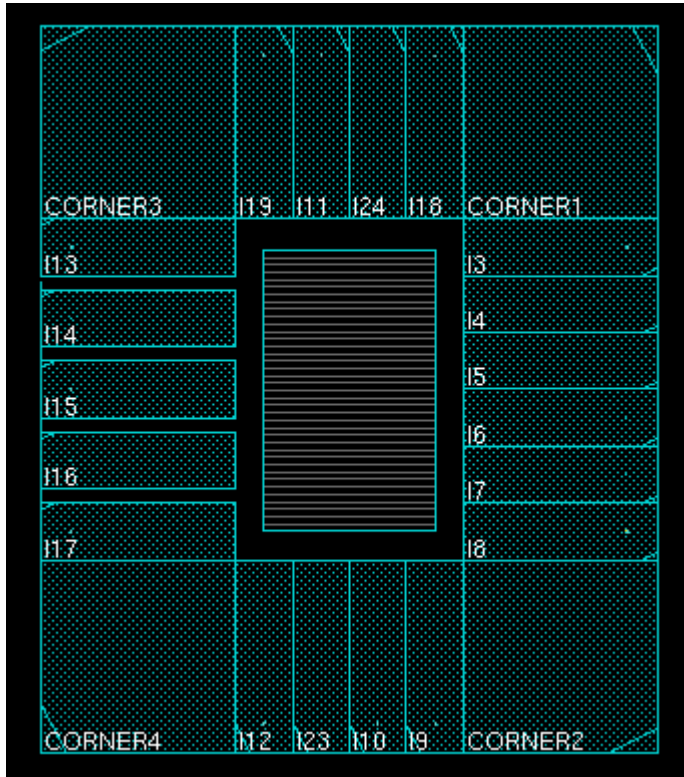


Fig: Floor Plan post amc 3 to 6 being run

12.8.2 Routing our selected Power Supplies

In order for the power supply connectivity to be done correctly we need to use vdd and gnd and not the gnd! and vdd! referenced in command 7 of the ha sequence.

Instead of typing amc 7 we need to type the following:

- `amsPowerRoute { {vdd 20} {gnd 20} }`

This should result in a design with the relevant power rails connected. The floorplan that should result is shown below:

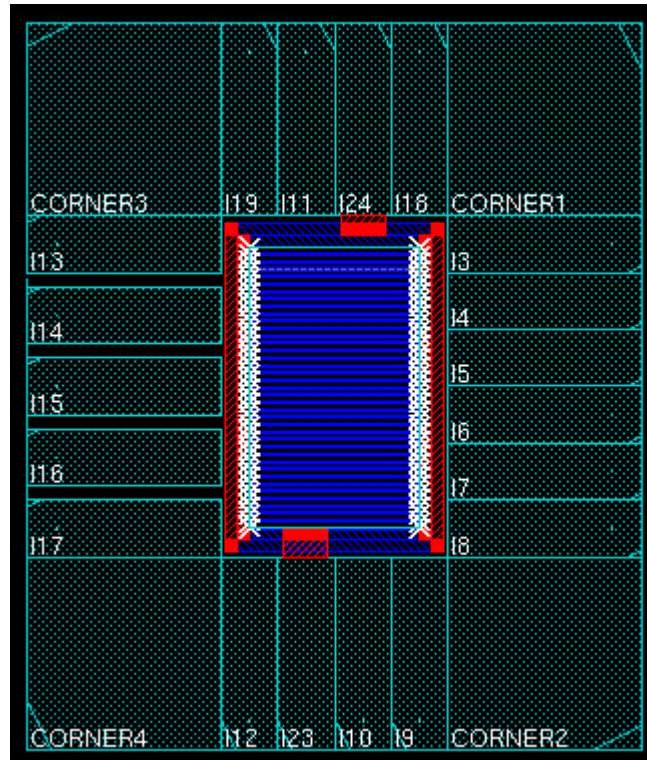


Fig: Design Showing Power Nets Connected by `amsPowerRoute { { gnd 20 } { vdd 20 } }` command

12.9 Pre and Final Routing Operations

We will now run the final operations from the script. Use the command

- `amc 8 9`

These will do the standard cell placement and ensure that the correct cells are placed at the end of each row to ensure that power supply connectivity is correct.

Now use the command

- `amc 13 16`

This will do the final chip routing and place filler cells where required, on the peripheral and core area. Again the chip structure has changed as can be seen below. We now have the relevant connections between the pads and the core as well as the power.

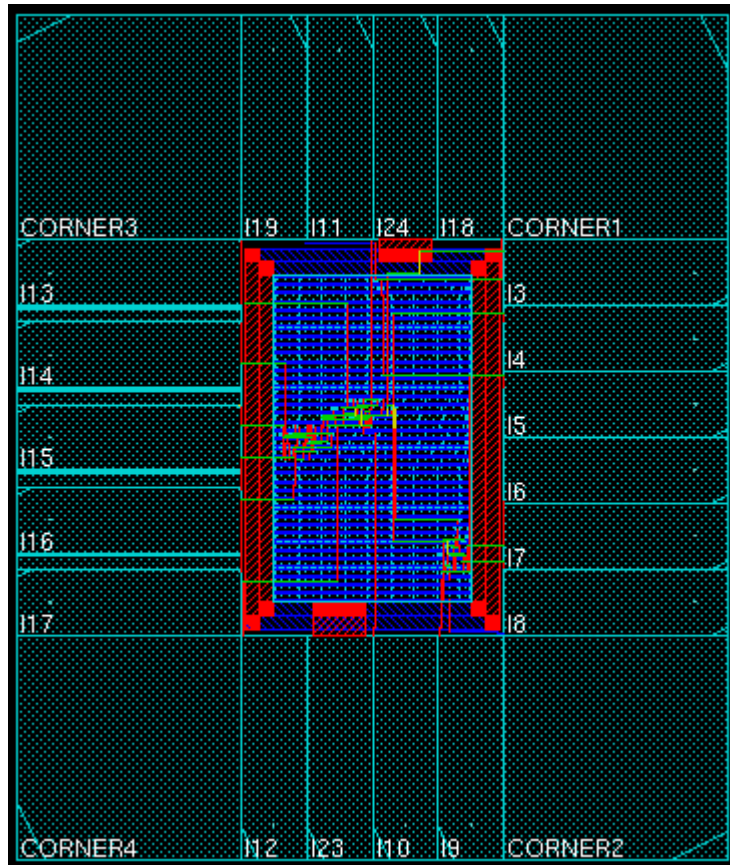


Fig: Post Final Route and Filler cell placement

12.10 Final Actions

It is probably a good idea to do a File -> Save Design at this stage.

Both for the encounter data type and the OA data type.

13 Caveats

As you can probably understand this is in no way an optimum design or placement for the overall requirements. I have utilised significant space on a core limited design to show the application of filler cells in both the peripheral and the core. Nor have we looked at CTS/PKS or a host of the facilities available within this tool.

However that said the final design should be a manufacturable ASIC conforming to our requirements. Congratulations.