***CMOS VLSI Design Course***

**LAB 1**

**Cell Design and Verification**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc48266626)

[1.1 Lab overview 1](#_Toc48266627)

[2 Learning Objectives 1](#_Toc48266628)

[3 Overview of VLSI CAD Tools 2](#_Toc48266629)

[4 Tool Setup 2](#_Toc48266630)

[5 Start Virtuoso 2](#_Toc48266631)

[5.1 Library Manager 3](#_Toc48266632)

[5.2 Create a library 3](#_Toc48266633)

[6 Part 1a: Schematic Entry 4](#_Toc48266634)

[6.1 Create schematic view 4](#_Toc48266635)

[6.2 Create component instances 5](#_Toc48266636)

[6.3 Useful shortcuts 6](#_Toc48266637)

[6.4 Create pins 6](#_Toc48266638)

[6.5 Create wires 6](#_Toc48266639)

[6.6 Save Schematic 7](#_Toc48266640)

[7 Part 1b: Logic Verification 7](#_Toc48266641)

[8 Part 1c: Schematic Simulation 10](#_Toc48266642)

[8.1 Launch NC-Verilog and netlist design 10](#_Toc48266643)

[8.2 Edit the generated files and create test vectors 11](#_Toc48266644)

[8.3 Simulate the design using NC-Verilog 11](#_Toc48266645)

[9 Part 1d: Create symbol 12](#_Toc48266646)

[9.1 Open Virtuoso Symbol Editor 12](#_Toc48266647)

[9.2 Modify symbol shape 12](#_Toc48266648)

[10 Part 1e: NOT Gate 13](#_Toc48266649)

[11 Part 1f: Hierarchical Schematic 14](#_Toc48266650)

[12 Part 2a: Layout 15](#_Toc48266651)

[12.1 Design rule 15](#_Toc48266652)

[12.2 Build the layout 19](#_Toc48266653)

[12.2.1 Familiarize with the Layout editor 20](#_Toc48266654)

[12.2.2 Power routing 21](#_Toc48266655)

[12.2.3 Draw n-active and p-active 21](#_Toc48266656)

[12.2.4 Draw poly gates 21](#_Toc48266657)

[12.2.5 Substrate and well taps 21](#_Toc48266658)

[12.2.6 n-well and select layers 22](#_Toc48266659)

[12.2.7 Routing grid 22](#_Toc48266660)

[12.2.8 Input and output pins 23](#_Toc48266661)

[12.3 Sanity check your design 23](#_Toc48266662)

[12.3.1 Run Design Rule Check 23](#_Toc48266663)

[12.3.2 LVS 23](#_Toc48266664)

[12.4 Cell Library Guidelines 26](#_Toc48266665)

[13 Part 2b: Hierarchical Layout 27](#_Toc48266666)

[13.1 What you should do 27](#_Toc48266667)

[14 For Independent Practice 28](#_Toc48266668)

[15 What to Turn In 29](#_Toc48266669)

# Introduction

## Lab overview

This is the first of four chip design labs for the CMOS VLSI Design course. The labs teach the practicalities of chip design using commercial CAD tools from Cadence and Synopsys. The first two labs emphasize the fundamentals of custom design, while the next two use logic synthesis and automatic placement to save time.

This lab introduces you to the basics of how to use Cadence to design, simulate, and verify schematics and layout of logic gates. It also serves as a stand-alone tutorial to quickly get up to speed with the Cadence tools. This lab is divided into two parts.

In the first part of this lab, you will draw transistor-level schematics to build cells for NAND, NOT, and AND gates and create a symbol for each cell. You will then simulate each cell by applying digital inputs and checking that the outputs match your expectations.

In the second part of this lab, you will draw a layout for each gate you created in part 1. The layout indicates how the transistors and wires are physically arranged on the chip. Using design rules check (DRC) and layout versus schematic check (LVS), which is provided by the tool, the layout is checked to ensure it satisfies the design rules and that the transistors match the schematic.

Finally, you will apply what you have learned by independently designing NOR and OR gates.

The next three labs extend this one to build the simple 8-bit microprocessor chip described in the Getting Started guide. Most of the microprocessor is provided, but one of each interesting piece has been removed. In Lab 2, you will use the AND and OR gates to complete an arithmetic/logic unit (ALU) and then finish the datapath. In Lab 3, you will design the aludecoder logic by hand and then synthesize, place, and route a controller. In Lab 4, you will place the controller, aludecoder, and datapath together, including the input/output pads and autoroute the whole design to form the complete chip.

# Learning Objectives

At the end of this lab, you should be able to:

* Draw transistor-level cell schematics using Cadence Virtuoso
* Simulate transistor-level schematics using NC-Verilog
* Draw cell symbols using Cadence Virtuoso
* Draw cell layouts using Cadence Virtuoso
* Perform design rule checking (DRC) and fix discrepancies
* Perform layout vs. schematic (LVS) checking and fix discrepancies
* Construct, simulate, and verify hierarchical cells

# Overview of VLSI CAD Tools

This set of laboratories uses the Cadence and Synopsys Electronic Design Automation (EDA) tools which when correctly set up are easy to use, as this tutorial will demonstrate.

There are two general strategies for chip design. Custom design involves specifying how every transistor is connected and physically arranged on the chip. Synthesized design involves describing the function of a digital chip in a hardware description language such as Verilog or VHDL and then using a computer-aided design tool to automatically generate a set of gates that perform this function, place the gates on the chip, and route the wires to connect the gates.

Most commercial designs are synthesized today because synthesis takes less engineering time. However, custom design gives more insight into how chips are built and into what to do when things go wrong. Custom design also offers higher performance, lower power, and smaller chip size.

# Tool Setup

These labs assume that your institution has the Cadence and Synopsys tools installed on a suitable Linux server as described in the Getting Started Guide. They also assume you have a way to connect to the server from a laptop or desktop computer and to run X Windows applications.

The tools generate a bunch of random files. It’s best to keep them in one place. In your home directory, create some directories by typing:

**mkdir IC\_CAD**

**mkdir IC\_CAD/cadence**

# Start Virtuoso

Before you start the Cadence tools, open a terminal and change into the cadence directory:

**cd ~/IC\_CAD/cadence**

Start Cadence with the NCSU extensions using the following command:

**cad-ncsu &**

|  |  |
| --- | --- |
| Lights On | The above command starts the Virtuoso software and will open the Virtuoso Design Environment and the Library Manager windows. |

A “What’s New” and a Library Manager window may open. You can **turn off** the *“What’s New”* window in the future by choosing **Edit • Off at Startup**.

You may see some warnings about “no route to host” and missing fonts in the terminal; you can safely ignore these. Scroll through the Virtuoso window and look at the messages displayed as the tool loads up. Get in the habit of watching for the messages and recognizing any that are out of the ordinary. This is very helpful when you encounter problems. At present, you shouldn’t see any warnings in Virtuoso.

## Library Manager

All of your designs are stored in a *library*. If the Library Browser doesn’t open, choose Tools • Library Manager. You’ll use the Library Manager to manipulate your libraries. Don’t try to move libraries around or rename them directly in Linux; there is some funny behavior, and you are likely to break them.[[1]](#footnote-2)

When you are all done, be sure to quit Cadence by choosing **File • Exit**… in the Virtuoso window. If you don’t, you may lose the work you’ve done and/or leave your library in a corrupted and unstable state. If this occurs, you may find a panic.log file in your home directory. The file will include directions to try to recover your work. Restart cad-ncsu. Before doing anything else, enter the instructions from panic.log into the Virtuoso window command line. For example:

dbOpenPanicCellView("lab3\_xx" "test" "schematic")

If Cadence crashes, it might leave your locks on open files in place. If your lock shouldn’t be there, you can manually remove it by going into the library directory and deleting the .cdslck files. As always, be careful when deleting files.

Familiarize yourself with the Library Manager. Your cds.lib file includes many libraries from the North Carolina State University Cadence Design Kit supporting the different MOSIS processes. It also includes libraries from the University of Utah. The File menu allows you to create new libraries and cells within a library, while the Edit menu allows you to copy, rename, delete, and change the access permissions.

## Create a library

Ensure the Library manger window is open and then create a library by doing the following:

* **Go to:** File • New • Library

|  |  |
| --- | --- |
| Lights On | If the Library Manager window is not open, you can open it from the Virtuoso window by selecting: **Tools • Library Manager**. |

* Name the library **lab1\_xx**, where xx are your initials.

|  |  |
| --- | --- |
| Lights On | Leave the path blank, and it will be put in your current working directory (~/IC\_CAD/cadence). Set the following options. |

* **Select:** “Attach to existing tech library”
* **Select:** UofU\_TechLib\_ami06 (also known as UofU AMI 0.60u C5N (3M, 2P, high-res)).

|  |  |
| --- | --- |
| Lights On | UofU\_TechLib\_ami06 is a technology file for the American Microsystems (now ON Semiconductor) 0.6 μm process, containing design rules for layout. |

You should see the new library **lab1\_xx** listed (left side) among other libraries in the Library Manager Window.

# Part 1a: Schematic Entry

In this part, you will create transistor-level schematic for a 2-input NAND gate. Each gate or larger component is called a cell. A cell can have multiple views such as schematic, layout, extracted, etc.

## Create schematic view

The schematic view for a cell built with CMOS transistors should be named **cmos\_sch**. Later, you will build a view called **layout** to specify how the cell will be physically manufactured.

To create a schematic view, in the Library Manager, **highlight** the library **lab1\_xx** and then:

* **Go to:** File • New • Cell View

In the New File window that opens:

* Confirm that **lab1\_xx** is selected for **Library**
* Enter **nand2** for **Cell** name
* Enter **cmos\_sch** for **View** name
* Enter **schematic** for **Type**
* Select **Schematic XL** for **Open with** and
* Ensure to **tick** the Checkbox **“Always use this application for this type of file”**
* Click *Ok*

|  |  |
| --- | --- |
| Lights On | You may get a window asking you to confirm that cmos\_sch should be associated with this tool (click Yes), or one complaining that it failed to check out the license for Virtuoso\_Schematic\_Editor\_L (Click Session). |

The schematic editor window will open.

Your goal is to draw a gate like the one shown in Figure 1. We are working in a 0.6 μm process with λ = 0.3 μm. Our NAND gate will use 12 λ (3.6 μm) nMOS and pMOS transistors.

To create the schematic as shown in Figure 1, you will need to create instances of transistors (nmos and pmos), power, and pins and connect them using wires.

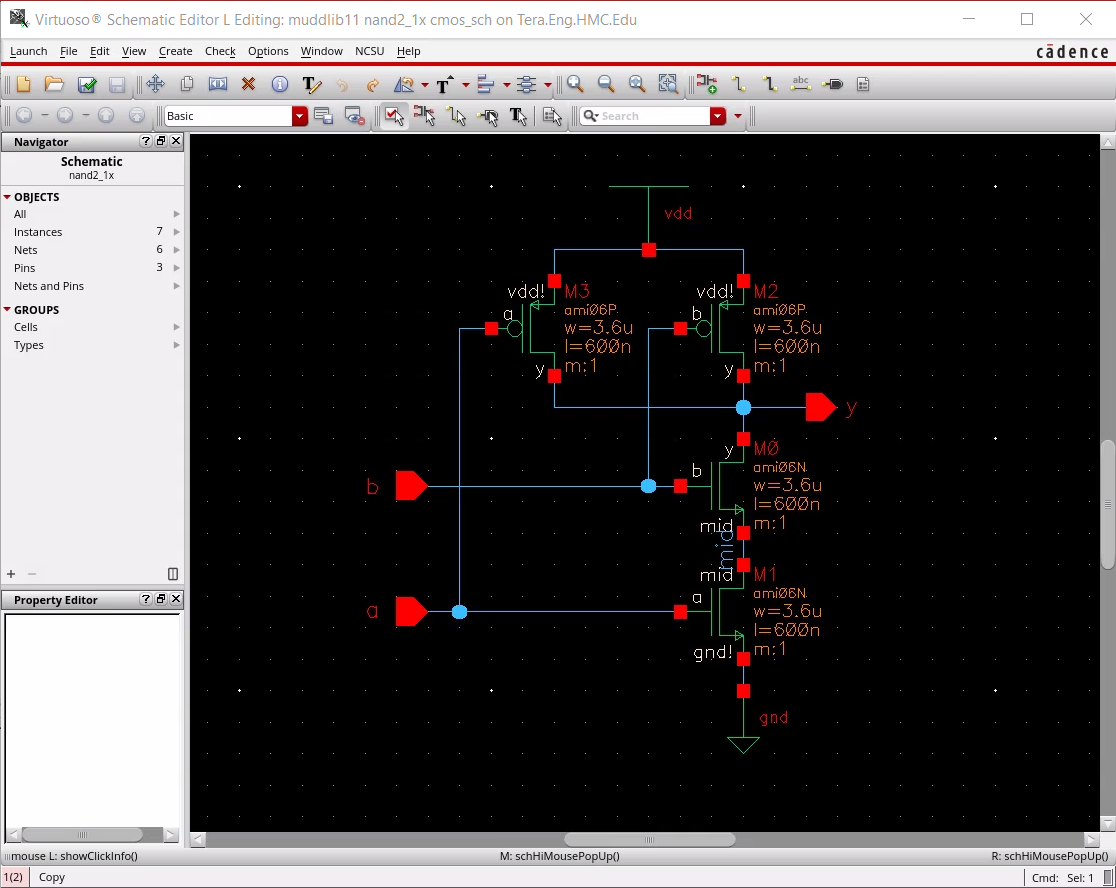


Figure 1: nand2 cmos\_sch

## Create component instances

Create transistor and power instances, by selecting transistors from the installed library using these steps:

* **Go to:** Create • Instance (i)

On the **Component Browser/Add Instance** window that opens:

* Click **Browse** and **Select:** UofU\_Analog\_Parts for the **Library**
* **Select:** **nmos** for the **Cell**
  + This expands the Add Instance window
* Confirm **symbol** is selected for **View**
* Set **Width** to **3.6u** (u indicates microns).
* **Click** in the schematic editor window to drop the transistor.
* You can **click a second time** to place another transistor.

Return to the Component Browser window and choose pmos. Set the width to 3.6u and drop two pMOS transistors in the schematic editor window. Also drop gnd and vdd in the schematic editor window.

Move the elements around until they are in attractive locations.

## Useful shortcuts

The following are shortcuts that will be useful when using Cadence tools.

* Exit a mode when in editor: *ctrl+c* OR *Esc* (press Esc twice if a dialog box is open)
* The bottom of the schematic editor window (see Figure 2) shows you the mode you are in. showClickInfo() is the normal mode
* Undo: *u* (note that saving clears your undo history)
* Copy: Edit • Copy (c)
* Move: Edit • Move (m)
* Delete: Edit • Delete (Del).
* Properties: Edit • Properties • Object (q)

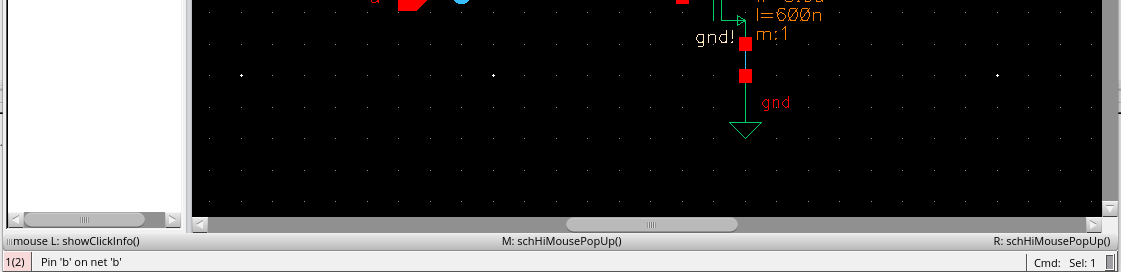


Figure 2: Mode Indicator Text

## Create pins

To create pins, do the following:

* Go to: Create • Pin… (Keyboard shortcut p).

On the Create Pin dialog box:

* Enter “a b” (no quotation marks, and a space between the two pin names).
* Set direction to “input.” The tools are case-sensitive, so use lower case everywhere.
* Place the pins, being sure that a is the bottom one. Pin order doesn’t matter logically, it does matter physically and electrically, so you will get errors if you reverse the order.
* Also place an output pin y.

## Create wires

To create wires and connect the components, do the following:

* Choose Create • Wire (narrow) (hotkey: w).
* Click on each component and click again to draw a wire to where it should connect.

|  |  |
| --- | --- |
| Lights On | It is a good idea to name every net (wire) in a design as this will aid in tracking down a problem later on one of the unnamed nets. Every net in your schematic is connected to a named pin or to power or ground except the net between the two series nMOS transistors. |

* Choose Create • Wire name… (l)
* Enter mid or something like that as the name
* Click on the wire to name it

|  |  |
| --- | --- |
| Lights On | It is unnecessary to label a wire connected to a named pin, power, or ground. If you do, it must have same name as the pin. The names of power and ground are vdd! and gnd! respectively. |

## Save Schematic

To save the schematic, Choose File • Check and Save to save your schematic.

You’ll probably get one warning about a “solder dot on crossover” at the 4-way junction on the output node. To stop this, go to

* Check • Rules Setup… and click on the Physical tab in the dialog.
* Change Solder On CrossOver from “warning” to “ignored” and close the dialog.
* Then, click *Check and Save* again and the warning should be gone.

Fix any other warnings. A common mistake is wires that look like they might touch but don’t actually connect. Delete the wire and redraw it.

Poke around the menus and familiarize yourself with the other capabilities of the schematic editor.

# Part 1b: Logic Verification

Cells are commonly described at three levels of abstraction. The register-transfer level (RTL) description is a Verilog or VHDL file specifying the behavior of the cell in terms of registers and combinational logic. It often serves as the specification of what the chip should do. The schematic illustrates how the cell is composed from transistors or other cells. The layout shows how the transistors or cells are physically arranged.

Logic verification involves proving that the cells perform the correct function. One way to do this is to simulate the cell and apply a set of 1’s and 0’s called test vectors to the inputs and then check that the outputs match expectation.

Typically, logic verification is done first on the RTL to check that the specification is correct. A testbench written in Verilog or VHDL automates the process of applying and checking all the vectors. The same test vectors are then applied to the schematic to check that the schematic matches the RTL.

You will begin by simulating an RTL description of the NAND gate to become familiar with reading RTL and understanding a testbench. In this tutorial, the RTL and testbench are written in System Verilog, which is a 2005 update to the popular Verilog hardware description language.

This tutorial describes how to use **NC-Verilog,** which integrates gracefully with the other Cadence tools. **NC- Verilog** compiles your Verilog into an executable program and runs it directly, making it faster than the older interpreted simulators.

Make a new directory for simulation (e.g., nand2sim). Copy nand2.sv, nand2.tv from the course directory into your new directory and copy nand2.testfixture, renaming it to testfixture.verilog. Use the commands below to do this:

**mkdir nand2sim**

**cd nand2sim**

**cp /courses/cmosvlsi/20/lab1/nand2.sv .**

**cp /courses/cmosvlsi/20/lab1/nand2.tv .**

**cp /courses/cmosvlsi/20/lab1/nand2.testfixture testfixture.verilog**

**nand2.sv** is the SystemVerilog RTL file, which includes a behavioral description of a nand2 module and a simple self-checking testbench that includes testfixture.verilog.

**testfixture.verilog** reads in testvectors from **nand2.tv** and applies them to pins of the nand2 module. After each cycle, it compares the output of the nand2 module to the expected output and prints an error if they do not match. Look over each of these files and understand how they work.

First, you will simulate the nand2 RTL to practice the process and ensure that the testbench works. Later, you will replace the behavioral nand2 module with one generated from your schematic and will resimulate to check that your schematic performs the correct function.

To invoke the simulator, on the command line, type:

**sim-nc nand2.sv**

You should see some messages ending with

ncsim> run

Completed 4 tests with 0 errors.

Simulation stopped via $stop(1) at time 81 NS + 0

You’ll be left at the ncsim command prompt. Type **quit** to finish the simulation.

If the simulation hadn’t run correctly, it would be helpful to be able to view the results.

**NC-Verilog** has a graphical user interface called SimVision. The GUI takes a few seconds to load, so you may prefer to run it only when you need to debug. To rerun the simulation with the GUI, type:

**sim-ncg nand2.sv**

A Console and Design Browser window will pop up as shown in Figure 3.

* In the browser, click on the **+** symbol beside the testbench to expand it
* Click on **dut**. The three signals, **a, b**, and **y** will appear in the pane to the right.
* Select all three and then **right-click** and choose **Send to Waveform Window**.

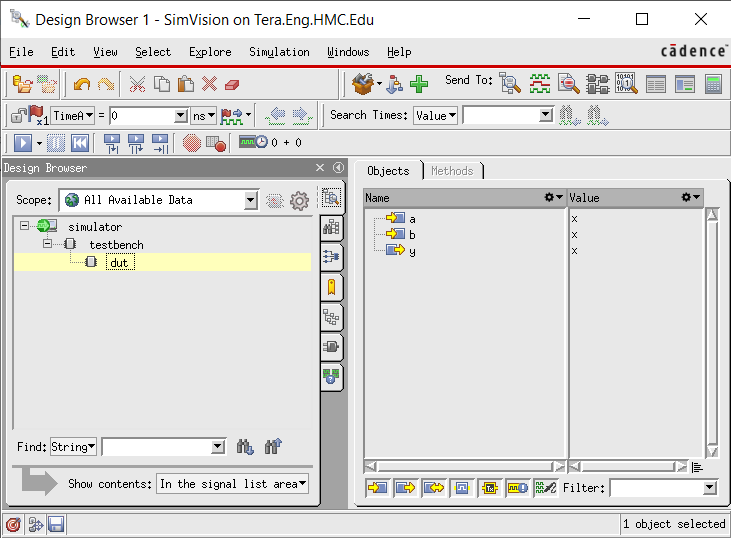


Figure 3: SimVision Design Browser

In the Waveform Window that opens (see Figure 4),

* Choose Simulation • Run.

|  |  |
| --- | --- |
| Lights On | You’ll see the waveforms of your simulation; inspect them to ensure they are correct. The 0 errors message should also appear in the console. |

If you needed to change something in your code or testbench or test vectors, or wanted to add other signals, do so and then **click** Simulation • Reinvoke Simulator to recompile everything and bring you back to the start. Then, choose Run again.

|  |  |
| --- | --- |
| Lights On | Make a habit of looking at the messages in the console window and learning what is normal. Warnings and errors should be taken seriously; they usually indicate real problems that will catch you later if you don’t fix them. |

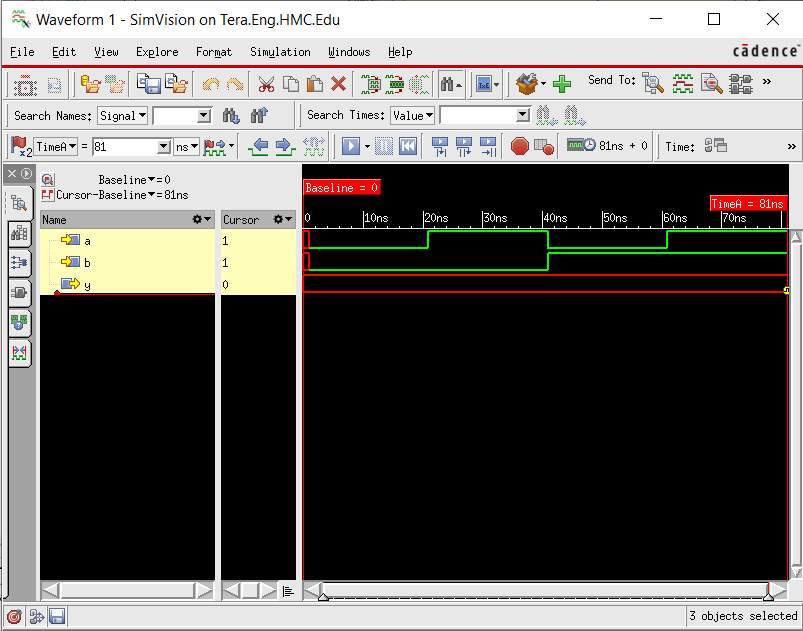


Figure 4: Successful nand2 simulation

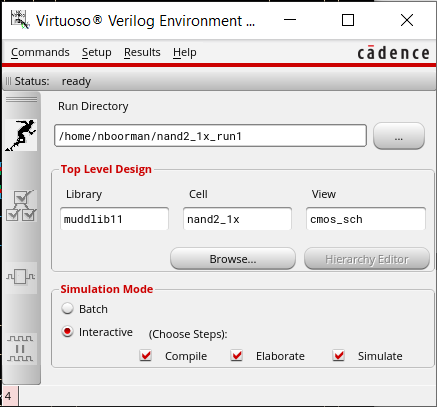
# Part 1c: Schematic Simulation

You will verify your schematic by generating a Verilog deck using the **NC-Verilog** tool and pasting it into the RTL Verilog file.

## Launch NC-Verilog and netlist design

To launch NC-Verilog, do the following

* On your schematic window, **Go to:** Launch • Plugins • Simulation • NC- Verilog
* Note the run directory in the Verilog environment that opens (shown in Figure 5).
* Press the button in the upper left to initialize the design. Virtuoso will create and initialize the directory for you.



Simulate

Generate Netlist

Initialize Design

Figure 5: NC-Verilog

* If a dialog pops up asking if you want to keep the previously run directory environment**, click No**.
* If a dialog pops up asking if you want to initialize simulation environment, **click OK**.

After initializing the design, on the NC-Verilog window:

* **Click** Setup • Netlist…; this opens a Netlist Setup window

In the newly opened window,

* **Check** the Netlist Explicitly option and click OK.

Back in the NC-Verilog window,

* Press the button with **three checkmarks** to **generate** a **netlist**.

Look in the Virtuoso window for errors and correct them. You should see that the pmos, nmos, and nand2 cells were all netlisted.

|  |  |
| --- | --- |
| Lights On | Beware that the netlister indicates “successful” even if it produces garbage! In the case that the netlister fails, try removing the directory created and redoing the process. |

## Edit the generated files and create test vectors

In your Linux terminal window, cd into the directory that was created. You’ll find quite a few files. The most important are **verilog.inpfiles, testfixture.template, and testfixture.verilog**.

Each cell is netlisted into a different directory under ihnl.

The file **verilog.inpfiles** states where they are. Take a look at the netlist and other files.

The file **testfixture.template** defines the top-level module that instantiates the device under test and invokes **testfixture.verilog**.

Run the following commands to copy your testfixture.verilog and test vectors from your nand2sim directory to your nand2\_run1 directory.

**cp ../nand2sim/testfixture.verilog .**

**cp ../nand2sim/nand2.tv .**

## Simulate the design using NC-Verilog

Back in the **NC-Verilog** window, you may wish to choose

* **Go to:** Setup • Record Signals.
* Select “All” in the window that opens to record signals at all levels of the hierarchy and click OK.

(This isn’t important for the nand with only one level of hierarchy but will be helpful later.)

* **Go to:** Setup • Simulation.
* Enter **simout.tmp –sv** for **Simulation Log File** and click **OK**.

|  |  |
| --- | --- |
| Lights On | This will print the results in simout.tmp. The **–sv** flag indicates that the simulator should accept SystemVerilog syntax used in the testfixture.verilog. |

Back in the **NC-Verilog** window

* **Set** the Simulator mode to **“Batch”** and click on the **Simulate** button.

|  |  |
| --- | --- |
| Lights On | You should get a message that the batch simulation succeeded. This doesn’t mean that it is correct, merely that it run. |

In the terminal window, view the simout.tmp file. It will give some statistics about the compilation and then should indicate that the 4 tests were completed with 0 errors.

If the simulation fails, the simout.tmp file will have clues about the problems. One common source of problem is when you renetlist after you have copied over testfixture.verilog, rewriting your testfixture with a default one that does nothing.

Change the simulator mode to Interactive to rerun with the GUI. Be patient; the GUI takes several seconds to start and gives no sign of life until then.

Add the waveforms again and run the simulation. You may need to zoom to fit all the waves. For some reason, SimVision doesn’t print the $display message about the simulation succeeding with no errors. You will have to read the simout.tmp file at the command line to verify that the test vectors passed. If you find any logic errors, correct the schematic and resimulate.

# Part 1d: Create symbol

Each schematic can have a corresponding symbol to represent the cell in a higher level schematic. In this step, you will create a symbol for your 2-input NAND gate.

When creating your symbol, it is a good idea to keep everything aligned to the grid; this will make connecting symbols simpler and cleaner when you need it for another cell.

## Open Virtuoso Symbol Editor

On your nand2 schematic editor window,

* **Go to:** Create • Cellview • From Cellview…
* Choose **cmos\_sch** for **From View Name**
* Choose **symbol** for **To View Name**
* Ensure that **Tool / Data Type** is set to **schematicSymbol** and **click OK**,
* Click Ok on the Symbol Generation Options window that will open.

Cadence will then create a generic symbol as shown in Figure 6.

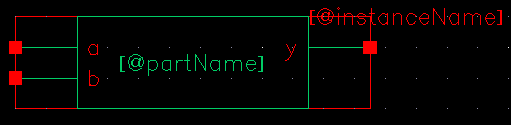


Figure 6: nand2 symbol

## Modify symbol shape

Next, modify this symbol to something familiar and easy to read as shown in Figure 7. Pay attention to the dimensions of the symbol; the overall design will look more readable when symbols are of consistent sizes.

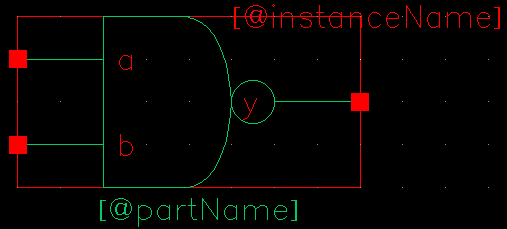


Figure 7: nand2 symbol final version

The green body of the NAND is formed from an open C-shaped polygon, a semicircle, and a small circle.

To form the semicircle,

* **Go to:** Create • Shape • Arc.

Experiment with the arc drawing tool.

To make the polygon

* **Go to:** Create • Shape • Line

To make the output bubble

* Create • Shape • Circle.

Move the lines and terminals around to make it pretty. The Edit • Stretch command may be helpful.

Finally,

* **Go to:** Create • Selection Box… and choose Automatic.

This creates a red box around the symbol that will define where to click to select the symbol when it appears in another schematic.

* Choose File • Check and Save when done.

# Part 1e: NOT Gate

Next, design a NOT gate named **inv**. Draw the cmos\_sch and the symbol, as shown in Figure 8. Make the pMOS width 10 λ (3u) and the nMOS width 7 λ (2.1u). Check and save when done.

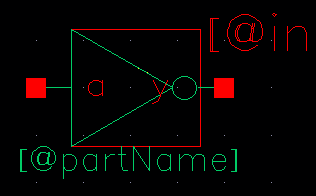
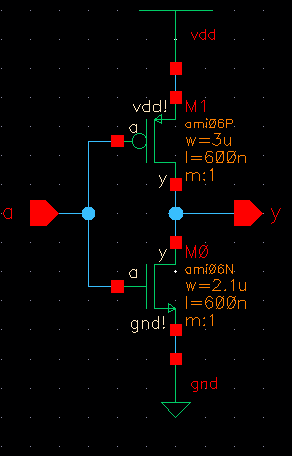


Figure 8: inv cmos\_sch and symbol

# Part 1f: Hierarchical Schematic

A CMOS AND gate consists of a NAND gate followed by a NOT gate. The schematic is constructed by connecting the symbols for the two gates that you have already drawn. This is an example of hierarchical design, reusing pre-existing components to save work.

* Create a schematic Cell named **and2** and enter schematic for the View.

|  |  |
| --- | --- |
| Lights On | This time, it is **schematic** and not cmos\_sch because the cell will instantiate other cells rather than transistors. |

* Add instances of **nand2** and **inv** from your **lab1\_xx** library.
* Wire the two gates together and **create** ports on inputs **a** and **b** and output **y** as shown in Figure 9. Name the wire between the two gates. In Figure 9, we named it **yb**.

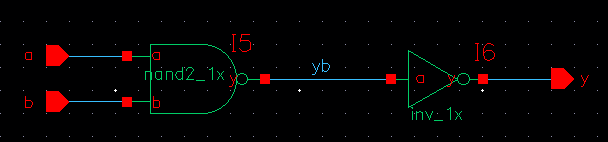


Figure 9: and2\_1x schematic

Simulate your and2 gate to ensure it works.

Copy the **nand2.tv** and **testfixture.verilog** files to the run directory for your and2 gate and modify them to contain the proper vectors for the and2 function.

|  |  |
| --- | --- |
| Lights On | If you encounter netlister errors about connectivityLastUpdated, be sure you have checked and saved the schematics of all of the components. |

Make a symbol for the and2. It should be similar to the nand2 but should leave off the output bubble. You may save yourself some time with judicious use of copy and paste or by using the copy command in the Library Manager.

# Part 2a: Layout

The next step is to draw the layout for the 2-input NAND gate. Figure 10 shows the stick diagram of the layout. Recall that where red crosses green, we get an nMOS transistor and where red crosses yellow, we get a pMOS transistor.

The nand2 has two **nMOS transistors** in **series** between **GND** and **Y**. It has two **pMOS** transistors in **parallel** between **VDD** and **Y**. The blue power and ground busses run horizontally in metal1. A bus means a wire in this context.

The **green n+ diffusion** (**ndiff**) runs **parallel** to and just **above** the **GND** bus. The **yellow p+ diffusion** (**pdiff**) runs **parallel** to and just **below** the **VDD** bus.

The **inputs** run vertically on polysilicon, crossing diffusion to form the four transistors. **Metal1** and contacts are used within the cell to connect the transistors to GND, VDD, and the output Y.

The transistors fit on the same pitch as the metal tracks, so the cell must be 3 tracks wide to accommodate the three contacts on the pMOS transistors.

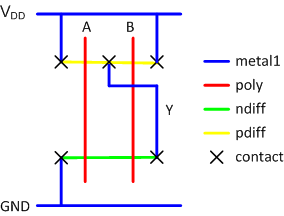


Figure 10: nand2 stick diagram

All cells should be drawn so that they “snap together” like LEGOs when placed adjacently. Therefore, we establish a set of rules for our cell library so they will satisfy design rules alone and also when connected together.

## Design rule

The first step in planning a cell is to consider the metal routing that will go overhead.

|  |  |
| --- | --- |
| Lights On | Note that in this specification, 1 λ = 0.3u (0.3 μm) |

In our system,

* metal2 will run vertically. Each contacted metal2 track has a pitch of 8 λ (4 λ width + 4 λ spacing to the next track).
* Metal3 will run horizontally. Each contacted metal3 track has a pitch of 10 λ (6 λ width + 4 λ spacing).
* To avoid wasted space, each cell should be a multiple of the track pitches in size. Hence, it should be a multiple of 8 λ wide and 10 λ tall.

All the cells in the same library need to be the same height so they can snap together. Thus, the cell height may be set by the tallest cell (generally a complex cell such as a flip-flop) with the widest transistors.

The cell height must also be tall enough to allow all of the horizontal wires in a datapath to pass overhead, which depends on the datapath being built. For the labs, we use a 10-track library with room for 10 horizontal metal3 lines to pass overhead. Based on the past experience, this is sufficient to easily build complex cells and to route interesting datapaths. Hence,

* the cells should be 10 tracks × 10 λ / track = **100 λ tall**.

The width of the cell depends on the connections inside the cell. Conveniently, the metal1 and metal2 pitch is the same (8 λ),

* from the nand2 stick diagram, the cell must be **3 tracks (24 λ) wide**.

Although metal1 can be 4 λ wide, we draw the GND and VDD busses 8 λ wide to carry the larger currents needed in the power supply.

In this context, there is ambiguity about *width* and *length* vs. *width* and *height*. We normally refer to a wire as having a length and width. The length is the long dimension and the width is the skinny dimension.

Thus, wires have a width of 8 and a length of 24 to run horizontally over the whole cell. But a rectangle has a width and a height, with the width being the x dimension and height being the y dimension. Thus, the width of the rectangles is 24 and the height is 8.

The origin of the cell is (x = 0; y = 0). Let us define x = 0 λ as the left side of the cell. Hence, x = 24 λ is the right side of the cell. See Figure 11.

Let us center the GND bus on y = 0. The bus has a width of 8, so it extends 4 above and below the center. Hence, the corners of the GND bus are (0, -4) and (24, 4). See Figure 11.

For a 10-track tall cell, the VDD bus must be 9 tracks higher, or centered at 90 λ. Hence, its corners are (0, 86 and 24, 94). See Figure 11.

|  |  |
| --- | --- |
| Figure 11: VDD and GND busses | Figure 12: Well and substrate contacts |

Each cell has **well** and **substrate** contacts centered under the VDD and GND busses, to tie the bodies of the transistors to the supply voltages. The pitch of the contacts is also 8 λ, so the number of contacts is the same as the number of tracks. These are shown in Figure 12.

The design rules call for a **spacing** of **4 λ** between **metal** and **diffusion**. To make cells snap together nicely, avoid putting any transistors or internal wires (excluding VDD and GND) closer than 2 λ to the left or right boundaries of the cell. Hence, adjacent cells will have a spacing of at least 4 λ between their internal contents.

To keep vertical metal 2 λ from the cell boundary, the leftmost vertical metal track should start at x = 2 and extend to x = 6. In other words, they are centered on x = 4. The track pitch is 8 λ, so the subsequent tracks are centered on x = 12, x = 20, etc.

Polysilicon has a width of 2 and is placed between the metal tracks to make transistors. Hence, it should be centered on x = 8, and x = 16. It has a minimum separation of 3 λ from unrelated diffusion. Hence, to avoid the substrate and well contacts, it must start 1 λ above GND and end 1 λ below VDD, as shown in Figure 13.

|  |  |
| --- | --- |
| Figure 13: Polysilicon inputs | Figure 14: ndiff and pdiff |

**Polysilicon** extends at least **2 λ** past diffusion when forming a transistor. Hence, the **ndiff** must start **3 λ above GND**. The **pdiff** must start **3 λ below VDD**. Like metal, beware of the ambiguity in the meaning of width of diffusion. Because poly is running vertically and diffusion is running horizontally, the width of a transistor corresponds to the height of the diffusion.

In our nand2 cell, the **heights** of the **ndiff** and **pdiff** are both **12 λ** because those are the widths of the nMOS and pMOS transistors in the schematic. In the schematic, we specified the width as 3.6u.

Recall that 1 λ = 0.3u (0.3 μm), so the height is 3.6u × 1 λ / 0.3u = 12 λ. Therefore, we need to draw strips of ndiff and pdiff starting and ending 2 λ from the ends of the cell (to give clearance to the next cell) and starting 2 λ above or below the polysilicon gates and extending vertically by 12 λ, as shown in Figure 14.

The **pMOS** transistors must be surrounded by an n-well by at least 6 λ on each side, and the well should be separated from the nMOS transistors by at least 6 λ. The well should be the same size in all cells to avoid interreference with nMOS in neighboring cells, and also to notch errors when cells are abutted. In our library, we choose to bring the n-well down to y = 40. This gives slightly more space for pMOS than nMOS transistors, which is a good choice because the pMOS are typically wider than nMOS. The n-well extends up to y = 96 to sufficiently enclose the well contacts. To surround transistors by 6, it must extend 4 λ left and right of the VDD bus, as shown in Figure 15.

|  |  |
| --- | --- |
| Figure 15: n-well | Figure 16: n-select and p-select |

The MOSIS design rules call for **n-select** and **p-select layers** enclosing the **ndiff and pdiff** by at least **2 λ**. Each is a rectangle. The n-select starts 1 λ above GND and ends 4 λ below the n-well. The p-select starts 1 λ below VDD and ends 4 λ above the n-well boundary. Both extend the full width of the GND/VDD metal lines, as shown in Figure 16. Hence, when cells are abutted, the active regions also abut, and there are no notches.

All leaf cells resemble Figure 16, with power and ground, n-well, well and substrate contacts, horizontal strips of diffusion, and vertical polysilicon inputs. Different cells will have different widths, but the template is otherwise the same.

The cells are customized by adding metal1 and contacts to connect the power, ground, and inputs and outputs. Most cells can be built using no metal2 or metal3 wires inside the cell, so wires can easily be routed over the cell in these layers without interference. Stick diagrams show poly running purely vertically, but we often bend the poly to bring transistors closer together if there is no contact required between them. This is worth the trouble because it reduces the capacitance, improving speed and power consumption.

## Build the layout

To start a layout of a 2-input NAND gate,

* First, **create a new cellview** for nand2 called **layout**. It should be of type layout and always open with **Layout GXL**.[[2]](#footnote-3)

|  |  |
| --- | --- |
| Lights On | Click OK to accept nand2 cmos\_sch when prompted about the Source View Definition. You may also get some warnings about Assura. |

Your goal is to draw a layout exactly like the one shown in Figure 17. Neatness and precision are imperative to get a good layout. Keep in mind the principles described above so the reasons for the dimensions are clear.

The University of Utah technology file is configured on a half-lambda grid, so grid units are 0.15 μm. Take care that everything you do is an integer multiple of λ so you don’t come to grief later on. The heavy dots are on a 10-λ grid and the smaller dots are on a 2-λ grid. **Also, units are listed in microns, so you will have to mentally convert 1 λ = 0.3 μm.**

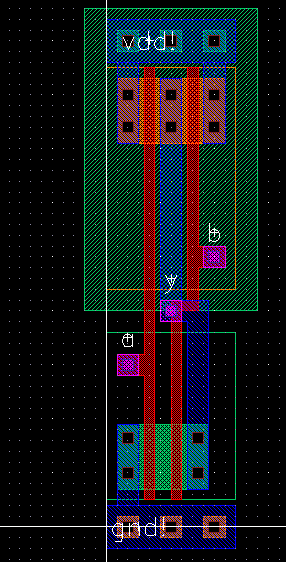


Figure 17: nand2 layout

### Familiarize with the Layout editor

The layers pane is on the left side of the Layout editor window. For the labs, you will need **nwell, nactive, pactive, nselect, pselect, poly, metal1, metal2, metal3, cc (contact cut), via, and via2.**

In this design process, nactive and pactive stand for n+ diffusion and p+ diffusion, respectively. They must be surrounded by a rectangle of nselect or pselect, respectively; in a cleaner flow, the select layers might be automatically generated.

By default, the Layout editor snaps to a 0.5 λ (0.15 μm) grid. Getting off the lambda grid will cause you grief, so start by changing this.

* Chose Options • Display… (*e*) and set the X and Y snap spacing to 0.3 (the units are microns).
* Set the major grid spacing to 3 microns (10 λ)
* Set the minor grid spacing to 0.3 micron (1 λ).

|  |  |
| --- | --- |
| Lights On | Using the save option, you may save your display settings to a file and reload it whenever you restart Cadence or the Layout editor using the load option. |

Some useful menu options when drawing the layout are:

* Zoom in and zoom out. There are several zoom commands under the window menu.
* Pan with the arrow keys.
* To create a rule for measurement, use Tools • Create Measurement (k).
  + Use the ruler to measure distances
  + Use Tools • Clear All Rulers to eliminate them when you are done.
* To stretch a drawing, use Edit • Stretch (s).
* To change the size or layer of a shape, use Edit • Properties (q)
* To merge multiple selected rectangles on a given layer into a single convenient polygon, use Edit • Basic • Merge (shift + m).

### Power routing

Start by placing your power and ground busses in metal1.

* Click on metal1 in the layers pane. **Choose** Create • Shape • Rectangle.
* Draw a rectangle from (0, -1.2) to (7.2, 1.2). This is a rectangle 8 λ wide and 24 λ long, centered on the y-axis with the left edge on the x-axis.
* Draw a second rectangle 90 λ (27 microns) above the first.

### Draw n-active and p-active

Next, draw the n-active and p-active. According to the schematic, the transistors are 12 λ wide, so the active rectangles should be 12 λ tall.

* Each should start 3 λ away from the ground or power bus.

### Draw poly gates

Next, draw two poly gates.

* They should be 2 λ wide and extend 2 λ beyond the transistors in each direction.
* The gate on the right must bend because the spacing between the series nMOS transistors is only 3 λ, while the spacing between the parallel pMOS transistors is 6 λ. You can make the bend by drawing the poly wire as three separate rectangles.
* Add metal 1 wires to connect the transistors to power, ground, and the output.
* Then, add 2 × 2 λ contacts (cc) between metal and the n-active.

In cells like this where the diffusion is wide enough, placing multiple contacts reduces the series resistance and increases reliability in case one contact is malformed during manufacturing. The exact placement is unimportant, however, each contact cut must be separated by 3 λ from its neighbors. You minimize resistance by placing as many contact cuts as possible and keeping the distance from any bit of diffusion to the contact as small as possible.

### Substrate and well taps

Recall that substrate and well taps are required to keep the diodes reverse biased. We will place taps under the power and ground wires on 8 λ centers.

* Draw three 4 × 4 λ squares of p-active underground, starting 2 λ from the edge.
* Place 2 × 2 λ contacts on the center of the taps to connect them to metal.
* Do the same with n-active under the power wire.

### n-well and select layers

Now is a good time to draw the n-well and the select layers.

* The n-well should extend from 40 λ to 96 λ vertically and should extend 4 λ beyond the edge of the cell in both directions (-4 to 28 λ).
* The p-select should extend from 44 to 85 λ vertically and should be as wide as the power line (0 to 24 λ).
* The n-select should extend from 5 to 36 λ vertically and should also be as wide as the power line.
* **Another** rectangle of n-select must be drawn exactly covering the power line to surround the n-well taps.
* A rectangle of p-select should be drawn covering the ground line to surround the substrate taps.

### Routing grid

Later, we will connect cells using vertical metal2 wires and horizontal metal3 wires.

* Metal2 is drawn on an 8 λ grid (width = 4 λ; spacing = 4 λ).
* Metal3 is drawn on a 10 λ grid (width = 6 λ; spacing = 4 λ).

Figure 18 shows the routing grid superimposed on the cell. The metal2 wires will run in the same columns as the well and substrate contacts, centered 4, 12, 20 λ, etc. right of the cell origin. The metal3 wires will run 0, 10, 20, 30, 40, 50, 60, 70, 80, and 90 λ up from the cell origin. Inputs and outputs should be placed on 4 × 4 λ squares of metal2 in the appropriate columns. For example, *a*, *b*, and *y* are all on this grid.

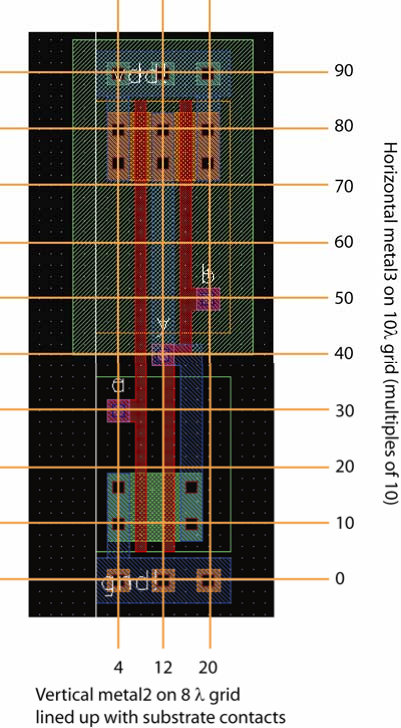


Figure 18: Wiring grids and pin locations

### Input and output pins

To create the inputs, you will need a stack going all the way from metal2 down to polysilicon: metal2, via, metal1, cc, and poly.

* The metal can be 4 × 4 λ, and the contacts/vias 2 × 2 λ, but the poly may need to extend a bit further to reach the main polysilicon lines.
* The output is already on metal1, so you only must add metal2 and a via.

Finally, place pins to define where the cell connects to other cells at the next level of the hierarchy. Select metal2 in the layer window.

* **Choose** Create • Pin…
  + Set the terminal name to a.
  + Make sure the following are chosen
    - Connectivity: weak
    - Pin Shape: rectangle
    - I/O Type: input.
  + Turn on Create Label.
  + Then, draw a 4 × 4 square of metal2 representing the pin on top of the metal2 already present for input a.
* Do the same for b.
* Do the same for y, but set it as an output.
* Also create large metal1 **inputOutput** pins called **vdd!** and **gnd!**. The pins should cover the entire power and ground busses. The “!! indicates a global net and is pronounced “bang.”

## Sanity check your design

Double-check that everything is aligned to a 1 λ grid and that the inputs and outputs are on the wiring grid. Ports should use all lowercase letters. Make sure you don’t have any 3 λ wide wires or any stray pins or other junk in the layout.

### Run Design Rule Check

* **Got to:** Verify • DRC… Click OK to run with the default settings (flat, don’t join nets with same name, divaDRC.rul, UofU\_TechLib\_ami06 rules library).
* Look at the Virtuoso window for errors. It is normal to have quite a few the first time you do layout, and all of them must be corrected.
  + Use Verify • Markers • Explain to zoom to the errors and get more information about them.
  + Use Verify • Markers • Delete All removes all the markers that may be cluttering the screen.

Your design should now resemble Figure 18 and should pass DRC. Save the layout.

### LVS

The next step is to prove that the layout matches the schematic. This is done by extracting the transistors and their connections from the layout and then running a layout-vs.-schematic (LVS) tool.

#### Extract transistors and connections

To extract the transistors and connection, in the layout editor,

* **Got to:** Verify • Extract… and run it with the default settings to create an **extracted** cellview. Make sure “Extract Method” is “flat.”

Look in the Virtuoso window for errors. You may wish to open the new view from the library manager to see what was produced.

#### Run LVS

To run LVS, in the layout editor,

* **Got to:** Verify • LVS… (opens a window similar to Figure 19)
* For the schematic input, select the **cmos\_sch** view of **nand2** in your **lab1\_xx** library.
* For the extracted input, choose the extracted view of the same cell and library.
* Select all four LVS Options (Rewiring, Device Fixing, Create Cross Reference, and Terminals).
* **Click Run** to launch the LVS job.

|  |  |
| --- | --- |
| Lights On | The default settings for LVS are good, but if they get cleared, LVS will mysteriously not work sometimes, so if you are sure a cell should be passing LVS and it doesn’t, check that the Run Directory is LVS, the Rules File is divaLVS.rul, and the Rules Library is checked and is UofU\_TechLib\_ami06. |



Figure 19: LVS settings

If you are lucky, you will get a window popping up saying that the job succeeded. This doesn’t mean that LVS passed, but merely that it ran. Click the Output button to look at the results. Familiarize yourself with the results. You want to see a statement that “**The net-lists match**.” Tracking mismatches is an acquired skill and can be tricky. Start with a careful visual examination of both the layout and schematic to be sure you’ve drawn what you intended. Often, looking in the LVS directory mentioned in the report will give more information about problems. This may involve a tedious effort of reading the netlist file and sketching a diagram of the mismatched components.

#### LVS debugging

If LVS fails, it can be difficult to track down the problem just from the output window alone. To make it easier, the Error Display view can try to give you an idea of where in the layout the error is. Although it is sometimes vague or incorrect, it still usually provides a good starting point.

To access the Error Display, go back to the library manager and open the Extracted view for the cell. This is the view that was generated by the Extract… command from Layout, so be sure to make your changes to the Layout view instead of this one.

As an example, I switched the a and b terminals in layout, which caused LVS to fail. The exacted view for the (incorrect) cell is shown in Figure 20.

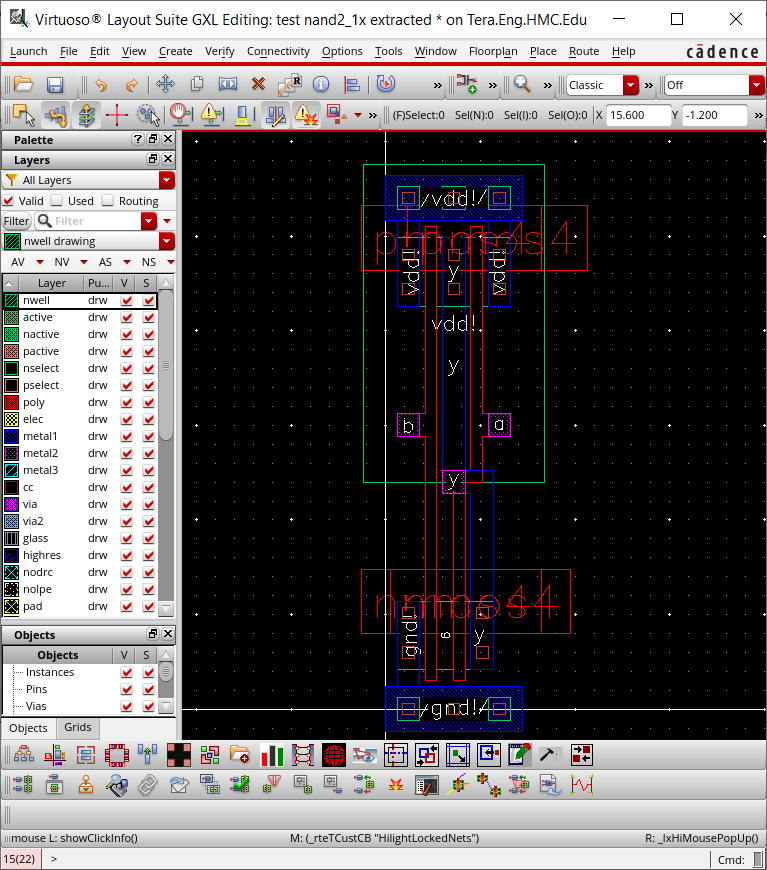


Figure 20: nand2 extracted view

Switch back to the LVS window (*not* the layout window) and click Error Display. A new window should open. It can be helpful to change the Error Color to white. Select the Auto-Zoom checkbox and click First. The Extracted view should zoom in and highlight approximately where the error might be, as seen in Figure 21. In this case, it detected that the nMOS transistors are in the wrong order, so it is highlighting one edge of them. The message in the Error Display window might also be useful.

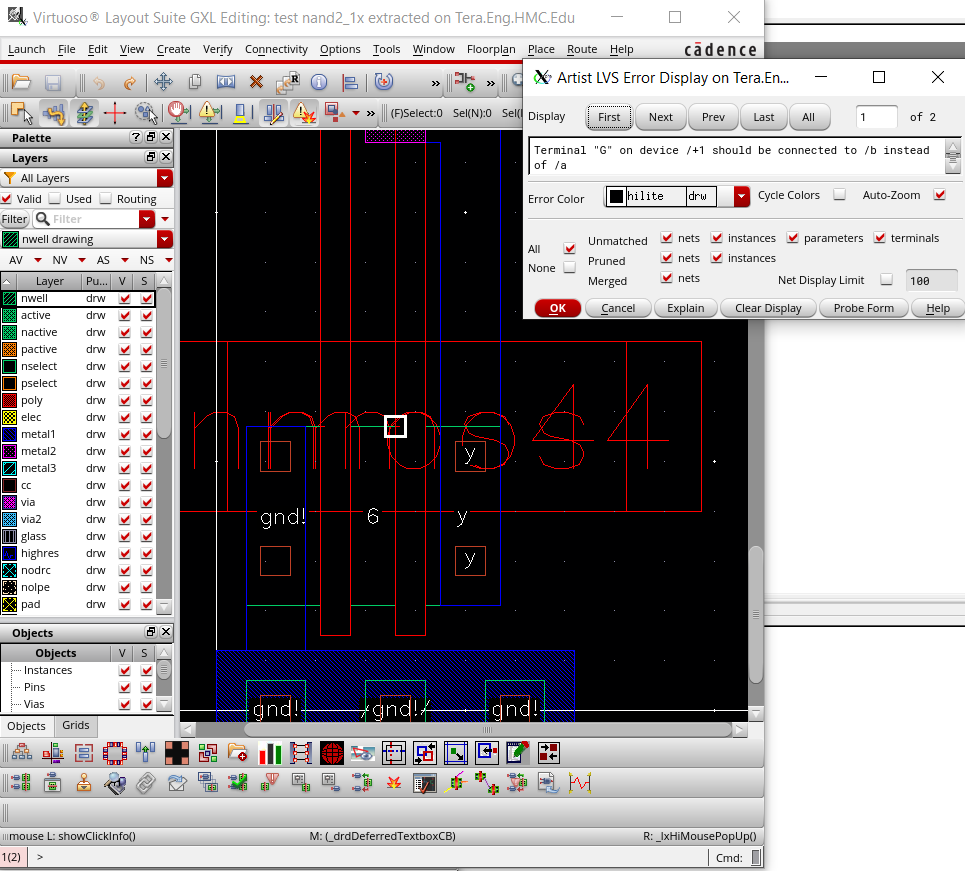


Figure 21: LVS Error Display

The Error Display can sometimes work on the schematic view, which is often easier to understand. Do not be discouraged by a high number of errors, as a single problem may cause dozens of errors to show up in this display.

## Cell Library Guidelines

Recall that we want our cells to snap together easily. Therefore, all gates should be 90 λ tall from the center of GND to the center of VDD. The width of the gate depends on the number of inputs. To match our routing grid, a logic gate is typically 8 λ wide for each input or output.

When you draw layouts, follow these guidelines:

• Place the nMOS transistors close to GND and the pMOS transistors close to power (with poly ending 1 λ away from the wide wire). The largest transistors that can be drawn are 27 λ for nMOS and 37 λ for pMOS.

• Neatness counts. Pack transistors as close as possible to minimize stray capacitance and resistance. Keep wires as short as possible.

• Place input and output pins on metal2. The pins should fall on the routing grid. There should never be more than one pin in a column.

# Part 2b: Hierarchical Layout

To illustrate hierarchical layout, we will build an AND gate using a NAND and an inverter. Figure 22 shows what the completed AND gate should look like. Observe how the cells abut nicely, with no notches in the n-well or active, and a spacing of at least 4 λ between the metal and diffusion within each subcell.

## What you should do

* First, draw the inverter layouts. Be sure that the nMOS and pMOS widths match the schematic. Check that it passes DRC and LVS.
* Next, create a new cell layout for the and2. Choose
  + Create • Instance and place the nand2 and inv layouts.
    - They show up as red bounding boxes.
    - Use Options • Display to look inside the box.
    - One option is to set the Stop Display Level to a big number (such as 10) to look inside cells.
    - But a more convenient option for us is to click on the Instance Pins box to show just the pins.
* Move the cells so that their power and ground busses abut.
* Place a 2 × 2 λ via2 on the **y** pin of the nand2 and on the **a** pin of the inv.
* Then, draw a metal3 wire connecting the two gates. Metal3 is thicker and has sloppier tolerances, so it must be 6 λ wide and extend 2 λ beyond each via2.

At this point, your design should resemble Figure 23.

|  |  |
| --- | --- |
| Figure 22: and2 layout | Figure 23: and2 hierarchical layout view |

Now you will need new pins at this level of hierarchy for a, b, y (from the inverter output), **vdd!** and **gnd!** The inputs and outputs should be 4 × 4 metal2 squares on top of the pins from the lower level of hierarchy, while the power and ground should be metal pins covering the entire power/ground busses.

Check your design with DRC and LVS and fix any errors.

# For Independent Practice

Now that you’ve learned the custom cell design flow, try it again yourself. Design schematics, symbols, and layouts for the following two gates. Simulate the schematics to prove that they work correctly. You will have to create your own System Verilog testbenches and test vector files. Check DRC and LVS. You will use your cells to complete the ALU in the next lab.

• 2-input NOR gate: nor2 (use transistor widths of 8 λ for the nMOS and 16 λ for the pMOS)

• 2-input OR gate: or2 (using a nor2 and an inv)

# What to Turn In

Please provide a hard copy of each of the following items. You may wish to use the Windows Snipping Tool or the Mac Grab program to take screenshots of your designs.

1. Please indicate how many hours you spent on this lab. This will not affect your grade, but will be helpful for calibrating the workload for the future.

2. A printout of your nor2 schematic.

3. A printout of your nor2 layout.

4. A printout of your or2 schematic.

5. A printout of your or2 layout.

6. A printout of your or2 test vectors. Does the schematic pass simulation?

7. Does the or2 layout pass DRC? LVS?

1. Copying a library with cp -r <srcpath> <dstpath> does work correctly, and you will find this helpful when working with a partner on the final project. [↑](#footnote-ref-2)
2. If you use Layout L instead of GXL, you may have LVS errors later. [↑](#footnote-ref-3)