***CMOS VLSI Design Course***

**LAB 4**

**Chip Assembly**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc48854857)

[1.1 Lab overview 1](#_Toc48854858)

[2 Learning Objectives 1](#_Toc48854859)

[3 Complete Chip 1](#_Toc48854860)

[3.1 Schematic 1](#_Toc48854861)

[3.2 Add Padframe and wire schematic 2](#_Toc48854862)

[3.3 Simulate chip 2](#_Toc48854863)

[3.4 Chip Layout 3](#_Toc48854864)

[3.4.1 Generate layout from source 3](#_Toc48854865)

[3.4.2 Position generated layout 3](#_Toc48854866)

[3.4.3 Manually Route power and ground 4](#_Toc48854867)

[3.4.4 Auto route connection 4](#_Toc48854868)

[3.4.5 DRC and LVS checks 5](#_Toc48854869)

[3.5 Tapeout 5](#_Toc48854870)

[3.6 Verify the GDS file 5](#_Toc48854871)

[3.6.1 Run DRC on imported layout 5](#_Toc48854872)

[3.6.2 Run LVS on imported layout 6](#_Toc48854873)

[4 What to Turn In 6](#_Toc48854874)

# Introduction

## Lab overview

In this final lab, you will assemble and simulate your entire microprocessor! You will build your top-level chip cell by connecting the datapath, aludecoder, and controller to a padframe cell containing the I/O pads. Chip will have the same inputs, outputs, and function as the top-level processor8 module.

The tiny transistors on a chip must eventually be attached to the external world with a padframe. A padframe consists of metal pads about 100 microns square; these pads are large enough to be attached to the package during manufacturing with thin gold bonding wires. Each pad also contains large transistors to drive the relatively enormous capacitances of the external environment.

You will first put together a schematic for the chip and simulate it to ensure the design is correct. You will then use the Virtuoso chip assembly router to automatically wire the chip layout according to your schematic. Finally, you will verify the layout and generate a GDS file suitable for manufacturing.

# Learning Objectives

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

* Assemble a chip schematic and layout including previously designed blocks and a padframe
* Use the Virtuoso router to connect components in the layout
* Verify the chip with simulation, DRC, and LVS
* Tape out the chip in GDS format and verify the GDS

# Complete Chip

## Schematic

Create a new schematic for a cell called chip in your processor8 library.

* Place symbols for the
* datapath and for
* aludecoder and
* controller from your controller\_xx library and
* wire the connections between these cells.

It is a good practice to place labels on the wires between the cells so that you will have an easier time debugging if problems arise.

If your aludecoder doesn’t need Funct bit 1, you’ll have to tap bits 4:2 and bit 0 off the bus.

## Add Padframe and wire schematic

The processor8 library contains a 40-pin padframe using pads from the UofU\_Pads library. Look at the padframe schematic and layout. If you were to build a chip with a different pinout, you would need to modify the padframe to put the proper types of pads (pad\_in, pad\_out, pad\_vdd, or pad\_gnd) in the desired positions.

The top-level inputs and outputs are listed in Table 1.

* Place a symbol for the padframe.
* Create pins for these inputs and outputs and connect them to the top of the padframe.
* Wire the \_core signals from the bottom of the padframe to the blocks within the chip.
* Again, name these internal wires.
* Check and save.

Table 1: chip Inputs and outputs

|  |  |
| --- | --- |
| **Inputs** | **Outputs** |
| ph1 | Adr<7:0> |
| ph2 | WriteData<7:0> |
| reset | MemWrite |
| ReadData<7:0> |  |

## Simulate chip

Simulate the chip with NC-Verilog.

* Generate a netlist in chip\_run1.

To simulate it using the same testbench as in Lab 2, you will need the external memory, the testbench, and the memfile.dat.

* Copy the testbench and memfile.dat to the run directory:

**cp /courses/cmosvlsi/20/lab2/processor\_multi.sv ~/IC\_CAD/cadence/chip\_run1**

**cp /courses/cmosvlsi/20/lab2/memfile.dat ~/IC\_CAD/cadence/chip\_run1**

* Open processor\_multi.sv in a text editor.
* Comment out all the modules from processor8 through the end, keeping only **testbench**, **mem**, and **top**.

Look at the testbench module. It instantiates the microprocessor as the device under test. You need to replace it with the netlisted schematic.

Look at the verilog.inpfiles file and find where chip was netlisted (e.g., ihnl/cds54/netlist).

* In **top**, comment out the processor8 instantiation and add a new instantiation of the chip using the ports in the proper order given in the chip netlist. For example:

//processor8 processor8(ph1, ph2, reset, MemWrite, Adr, WriteData, ReadData);

chip c(Adr, MemWrite, WriteData, ReadDta, ph1, ph2, reset);

* As in lab 2, invoke the simulation with the following command:

**sim-nc processor\_multi.sv –f verilog.inpfiles**

Look for a “Simulation succeeded” message. If the simulation doesn’t terminate within a few seconds, it probably has an error and will never meet the completion condition. If the simulation is unsuccessful, fire up sim-ncg, add some interesting waveforms, and systematically diagnose the problem. You may find it helpful to compare against the known good waveforms from Lab 2. The most likely places for mistakes are in your routing between modules in chip.

## Chip Layout

In this step, you will use the Virtuoso router to auto-route the chip layout based on the connections specified in the schematic.

* Open the chip schematic.
* **Go to:** Launch • Layout GXL.
* Click OK to create a new chip layout cellview.

### Generate layout from source

In the new layout window,

* **Go to:** Connectivity • Generate • All From Source…
* In the Layout Generation Options window, set the I/O pin default layer to metal2 and the width and height to 1.2 (microns).
* Click Apply to apply these defaults to all the pins.
* Uncheck the create box for vdd! and gnd! (in the Specify Pins to be Generated).
* In Pin Label click Create Label as Label.
* Then, navigate to the PR Boundary tab. The chip will be 1500 microns on a side. Under Area Estimation, change from Utilization to Width and set the width to 1600 (microns) to leave some slop around the edges. Then, choose OK.

### Position generated layout

You’ll see a purple place and route boundary box in the layout window, along with the four cells scattered outside the box. If you zoom in near the origin, you’ll also see the pins for all the chip ports. Set the display options so that you can see the contents of the cells.

* Move the padframe inside the place and route boundary. All of the pads should be within the boundary, though the labels with the pin numbers will extend outside.
* Move the other three cells (datapath, controller, and aludecoder) inside the padframe and arrange them with the datapath below the other two. Place them far enough apart that the router will be able to run wires between the cells.
* Find all of the pins (near the origin) and delete them all.

### Manually Route power and ground

The router doesn’t handle power and ground connections. The connections need to be beefy to handle the current drawn from the supply.

* Use some fat wires (e.g., 9.9 microns) and plenty of vias to manually connect power and ground. Pin 40 is gnd! and pin 39 is vdd! in the padframe. These should connect to the power/ground rings of the datapath and controller using beefy wires and plenty of vias.

The padframe is made of metal1 so you can connect metal1 wires to it by abutting them against the padframe (see Figure 1).

* Use some regular wires (e.g., 8 λ) to connect the aludecoder supplies because this module is fairly small.

Be sure not to mix up power and ground!

* Save a backup copy of this version in case your subsequent routing fails and you need to try again.

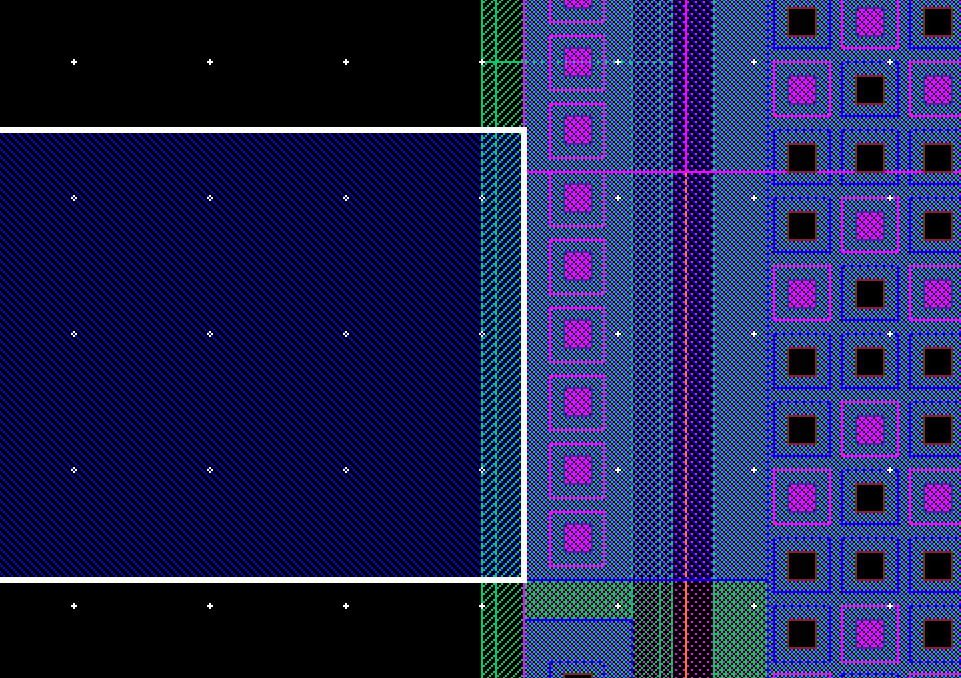


Figure 1: metal1 wire connection to padframe

### Auto route connection

* **Got to:** Route • Automatic Routing • Options… and deselect Exclude Blocked Pins During Routing.
* **Go to:** Route • Automatic Routing • All Nets. This should connect all four cells.
* If automatic routing did not work well,
  + **Go to:** Route • Delete Routing • All Nets can undo it.

### DRC and LVS checks

Run DRC and LVS on the routed chip. Make sure you **do not join nets with the same name**; this could hide a missing wire in the chip. The router occasionally introduces minor DRC problems that you may need to fix by hand.

## Tapeout

The final step in designing a chip is creating a file containing the geometry needed by the vendor to manufacture masks.

The two popular output formats are CIF (historically significant and human readable) and GDS (a binary format used in commercial designs today); we will use GDS (the Graphic Data System format).

To write a GDS file, in the Virtuos window,

* **Go to:** File • Export • Stream…
* Enter your library name (processor8), top cell name (chip), and view name (layout).
* Set the output file to chip.gds.
* Enter /proj/ncsu/rel/pipo/stream4gds.map as the Layer Map.
* Hit translate and view the log. (Look at this file and see how it maps the Cadence layers to 3-letter GDS layer names).
* Check for and resolve any errors. You may ignore the warnings about the layer map containing unknown layers such as metal4 that aren’t actually used in our process.

There should be no ellipses in the PIPO.log.

## Verify the GDS file

Verify that the GDS file is valid by reading it back into a new library.

* Create a new library named processor8\_gdsin. Be sure to attach the UofU Technology library.
* In the Virtuoso window, go to File • Import • Stream.
* Set chip.gds as the input file and chip as the top cell.
* Specify the new library (processor8\_gdsin) so that you don’t overwrite your chip.
* Load the map file from /proj/ncsu/rel/pipo/stream4gds.map.

You may ignore warnings about being unable to open the technology file. You can also ignore a fatal error about a loop in the hierarchy.

### Run DRC on imported layout

Run DRC on the imported layout. When opening your layout, you may get prompted to update the connectivity reference with a schematic view. You can continue without adding a reference to a schematic. You should see 144 DRC errors related to optional rule 10.4 about spacing from the pad to unrelated metal. (You do not get these errors on the original padframe because it was marked with a special “nodrc” layer.) You might also get a small number of errors about improperly formed shapes. If you do, use Verify • Markers • Find to walk through the errors until you find the bad shape and make sure it is not important. For example, there might be an improperly shaped piece of metal 2 overlapping an existing metal2 square for a contact; the bad portion can be ignored because of the overlap.

### Run LVS on imported layout

Extract the chip layout from processor8\_gdsin and run LVS to compare it against the chip schematic from processor8. The netlists should match although there will be no pins in the layout. Fix any problems.

This completes the lab. You now know how to create layouts and schematics. You know how to draw leaf cells and then build up custom datapaths and synthesized control logic blocks. You know how to verify the design using DRC, LVS, and simulation. And you know how to put the design into a padframe and run final checks before manufacturing.

May you build many interesting chips!

# What to Turn In

Please provide a hard copy of each of the following items:

1. Please indicate how many hours you spent in this lab. This will not affect your grade but will be helpful for calibrating the workload for the future.
2. A printout of the chip schematic.
3. Does the chip schematic simulate correctly?
4. Does the chip layout pass DRC? LVS?
5. A printout of the chip layout.
6. Does the imported GDS pass DRC? LVS?