CMOS VLSI Design
Lab 4: Full Chip Assembly

In this final lab, you will assemble and simulate your entire MIPS microprocessor! You will build your top level chip cell by connecting the datapath, aludec, and controller_synth to a padframe cell containing the I/O pads. chip will have the same inputs, outputs, and function as the top level mips 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.

I. chip Schematic

Create a new schematic for a cell called chip in your mips8 library. Place symbols for the datapath and for aludec and controller_synth from your `controller_xx`library, and wire the connections between these cells. It is 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 aludec doesn't need bits 5:4, you'll have to tap bits 3:0 off of the bus.

The `mips8` 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.

Inputs	Outputs
ph1	adr<7:0>
ph2	writedata<7:0>
reset	memread
memdata<7:0>	memwrite

Table 1: MIPS Processor Inputs & Outputs

Simulate the chip with NC-Verilog. Generate a netlist in `chip_run1`. To simulate it using the same test bench 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/e158/15/lab2/mips.sv ~/IC_CAD/cadence/chip_run1
cp /courses/e158/15/lab2/memfile.dat ~/IC_CAD/cadence/chip_run1
```

Open `mips.sv` in a text editor. Comment out all the modules from mips through the end, keeping only testbench and exmemory. Then look at the testbench module. It instantiates the mips processor 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 the testbench, comment out the mips instantiation and add a new instantiation of the chip using the ports in the proper order given in the chip netlist. For example:

```
//mips #(WIDTH,REGBITS) dut(.*);
chip c(adr, memread, memwrite, writedata, memdata, ph1, ph2, reset);
```

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

```
sim-nc mips.sv –f verilog.inpfiles
```

and look for a "Simulation completed successfully" 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.

II. chip Layout

In this step, you will use the Virtuoso Chip Assembly Router (VCAR) to autoroute the chip layout based on the connections specified in the schematic.

Open the chip schematic. Choose Launch • Layout XL. Click OK to create a new chip layout cellview.

In the new layout window, choose 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.

You'll see a purple place & 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 & route boundary. All of the pads should be within the boundary, though the labels with the pin numbers will extend outside. Then move the other three cells (datapath, controller_syn, and aludec) 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.

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_syn using beefy wires and plenty of vias. Use some regular wires (e.g. 8 λ) to connect the aludec 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.

Next, you will autoroute the signals. Choose Routing • Export to Router… Check the Use Rules File and set it to `/courses/e158/15/lab4/icc.rul`. The router will take a moment to start. It will over your terminal window and report some status.

To configure some more routing rules, choose File • Execute Do File…in the VCAR window and enter `/courses/e158/15/lab4/do.do`. The contents of the .do file are printed in the console.

Choose Autoroute • Global Route • Local Layer Direction. Click to get the Layer Direction from Layer Panel. This sets up routing with metal2 vertical and metals 1 and 3 horizontal. Next, choose Autoroute • Global Route • Global Router… The router will plan the approximate path for each wire. Ignore a possible warning that this is not a chip assembly application. Next, choose Autoroute • Detail Route • Detail Router. The detailed router does the routing. Check that it completes all of the connections (indicating 0 Unroutes). Make sure that you see wires connecting each of the cells within the core and also wire connecting the core to the pads. Choose Autoroute • Clean… to improve the routing. Finally, choose Autoroute • Post Route • Remove Notches to fix any notches left by the router.

Chose File • Write • Session and click OK. Then quit the router. The chip layout should be automatically updated with the new routes. If it isn't, use Routing • Import from Router… to import the session back into the layout.

Run DRC and LVS on the 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.

III. Tapeout

The final step in designing a chip is creating a file containing the geometry needed by the vendor to manufacture masks. Once upon a time these files were written to magnetic tape, and the process is still known as tapeout. Before taping out, run the checks mentioned above.

The two popular output formats are CIF and GDS; we will use GDS (the Graphic Data System format).

To write a GDS file, choose File • Export • Stream… in the Virtuoso window. Enter your library name (`mips8`), top cell name (chip), and view name (layout). Set the output file to `chip.gds`. Click on User-Defined Data and enter `/proj/ncsu/rel/pipo/stream4gds.map` as the Layer Map table. 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 labels or ellipses in the PIPO.log.

Look at the GDS file in a text editor. You should be able to identify the various cells. Each cell contains boxes (B) (rectangles) for each layer (L). For example, the CMF layer is first-level metal and the CWN is n-well. The C statement instantiates a cell whose number has already been defined in the file.

Verify that the GDS file is valid by reading it back in to a new library. Create a new library named `mips8_gdsin`. Be sure to attach the UofU Technology library. Choose File • Import • Stream in the Virtuoso window. Set `chip.gds` as the input file and chip as the top cell. Specify the new library (`mips8_gdsin`) so that you don't overwrite your chip. Attach UofU_TechLib_ami06. Then hit more options, layers and 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 the imported layout. 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.

Extract the chip layout from `mips8_gdsin` and run LVS to compare it against the chip schematic from `mips8`. 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, 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!

V. What to Turn In

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

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 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?

Modified in 2015 by Avi Thaker, Austin Fikes and David Money Harris, to support Virtuoso 6.1.5 and auto-routing.