CMOS VLSI Design
Lab 4: Full Chip Assembly

In this final lab, you will assemble and simulate your entire MIPS microprocessor! Overall, you will be working with three cells: core, padframe, and chip. core contains all of the logic for your chip. padframe contains the I/O pads. chip contains the core routed to padframe. Both core and chip should have the same inputs, outputs, and function as the top level mips module.

You will put together the datapath, aludec, and controller_synth to build your core. You will simulate the core to ensure your design is correct. You will then use the Virtuoso chip assembly router to automatically wire these modules up according to your schematic. Then you will connect the core to the padframe and again use the chip assembly router to make the connections.

FOR 2010 ONLY

The `mips8` library used in Lab 2 this year was missing the padframe and missing power rings on the datapath. Delete the old `mips8` and replace it with a new one:

```
rm –r ~/IC_CAD/cadence/mips8
cp –r /courses/e158/10/lab2/mips8 ~/IC_CAD/cadence
```

I. Core Schematic

Create a new schematic for a cell called core. Look at the symbol, which defines the inputs and outputs, which are also listed below.

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

Table 1: MIPS Processor Inputs & Outputs

Create pins for these same inputs and outputs. Place symbols for the datapath and for aludec and controller_synth from your `lab3_xx` library. Wire the cells together. 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. Check and save.

Simulate the core with NC-Verilog. Generate a netlist in `core_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/e158/10/lab2/mips.sv ~/IC_CAD/cadence/core_run1
cp /courses/e158/10/lab2/memfile.dat ~/IC_CAD/cadence/core_run1
```

Now, you will need to modify `testfixture.template` and `testfixture.verilog` to incorporate the external memory and testbench.

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 core was netlisted (e.g. `ihnl/cds47/netlist`). In the testbench, comment out the mips instantiation and add a new instantiation of the core using the ports in the proper order given in the core netlist. For example:

```
//mips #(WIDTH,REGBITS) dut(.*);
core (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.

II. Core Layout

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

Open the core schematic. Choose Tools • Design Synthesis • Layout XL. Click OK to create a new core layout cellview.

In the new layout window, choose Design • Gen 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. Set Pin Label Shape to Label. Under Boundary Area Estimation, change from Utilization to Boundary Width and set the width to 900 (microns). Then set the Aspect Ratio to 1 so the core of the chip will be placed in a 900 x 900 micron (3000 x 3000 λ) box. Then choose OK.

You'll see a purple place & route boundary box in the layout window, along with the three cells scattered outside the box. If you zoom in near the origin, you'll also see the pins for all the core ports.

Move each of the cells into the place & route boundary. Place them far enough apart that the router will be able to run wires between the cells.

Choose Place • Pin Placement… to move the pins into more sensible positions near where they come out of the core. Specifically, set adr, memdata, and writedata to the left side. Place ph1, ph2, reset, memread, and memwrite on the top. And set vdd! and gnd! on the bottom. Click Close to complete the placement. Zoom in and look around the edge of the placement boundary to check that the pins are distributed.

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) to connect the controller and datapath. If you need to connect different metal layers, use a generous supply of vias. Use some regular wires (e.g. 8 λ) to connect the aludec supplies because this module is fairly small.

Next, you will autoroute the signals. Choose Routing • Export to Router… Check the Use Rules File and set it to `/courses/e158/10/lab4/icc.rul`. The router will take 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/e158/10/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. Next, choose Autoroute • Detail Route • Detail Router.. The detailed router does the routing. Check that it completes all of the connections (indicating 0 Unroutes). 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 core layout should be automatically updated with the new routes.

Run DRC and LVS on the core. The router occasionally introduces minor DRC problems that you may need to fix by hand.

III. Chip Assembly

The tiny transistors on a chip must eventually be attached to the external world with a pad frame. A pad frame 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.

The mips8 library contains a 40-pin padframe using pads from the UofU_Pads library. Look at the 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.

Next, look at the chip schematic. The padframe has already been connected to the core.

Use VCAR to autoroute the padframe to the core. The chip will be 1500 microns on a side, so you may wish to set the boundary to 1600. You will need to manually connect one of the power and one of the ground pins to the core using beefy wires and plenty of vias. For example, pin 34 is VDD and pin 35 is GND. There will be no pins to arrange because the chip is the top level, so find the pins and delete them all.

Run DRC and LVS on the chip. LVS should match but show zero pins in the layout. If this were a real design, you would also simulate the chip schematic.

IV. 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 CIF (the Caltech Interchange Format) because it is a human-readable text file and thus easier to inspect for problems than the binary GDS format.

To write a CIF file, choose File • Export • CIF… in the icfb window. Enter your library name (mips8), top cell name (chip), and view name (layout). Set the output file to chip.cif. Click on User-Defined Data and enter /proj/ncsu/rel/pipo/cifOutLayermap as the Layer Map table. (Look at this file and see how it maps the Cadence layers to 3-letter CIF 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 CIF 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 CIF file is valid by reading it back in to a new library. Create a new library named mips8_cifin. Be sure to attach the UofU Technology library. Choose File • Import • CIF in the icfb window. Set chip.cif as the input file and chip as the top cell. Specify the new library (mips8_cifin) so that you don't overwrite your chip.

Choose `/proj/ncsu/rel/pipo/cifInLayermap` for the layer map table. You may ignore warnings about being unable to open the technology file.

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_cifin` 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 core schematic.
3. Does the core schematic simulate correctly?
4. Does the core layout pass DRC? LVS?
5. A printout of the chip layout.
6. Does the chip layout pass DRC? LVS?
7. Does the imported CIF pass DRC? LVS?