ECE 3450: Digital Electronics
Fall 2009, Lab #5

Dr. Pallav Gupta
pallav.gupta@villanova.edu

I Objectives

The goals of this lab are the following:

1. Assemble and layout the entire 8-bit MIPS processor.
2. Generate the pad frame and connect the "core" to the external world via I/O pads.
3. Become proficient in editing relatively large-scale designs.

II Courses Forum

The forum is located at http://pandim.ece.villanova.edu/phpbbforum. Please use it to ask questions.

III Submission Instructions

The lab must be submitted via the web at http://pandim.ece.villanova.edu. Instructions on how to submit are on the courses forum. Apply the following information on the Remository form when submitting your lab.

Filename:	`lab5-[first name]-[last name].zip`, *e.g.*, `lab5-john-doe.zip`
Location:	`ECE 3450/LAB5`
Title:	`Lab 5`
Author:	Your name, *e.g.*, `John Doe`

Note: Follow the above convention strictly. Failure to do so will result in a **zero**. We request your lab in this specific format so that the automated scripts at the back-end can run smoothly without breaking. Please adhere to it.

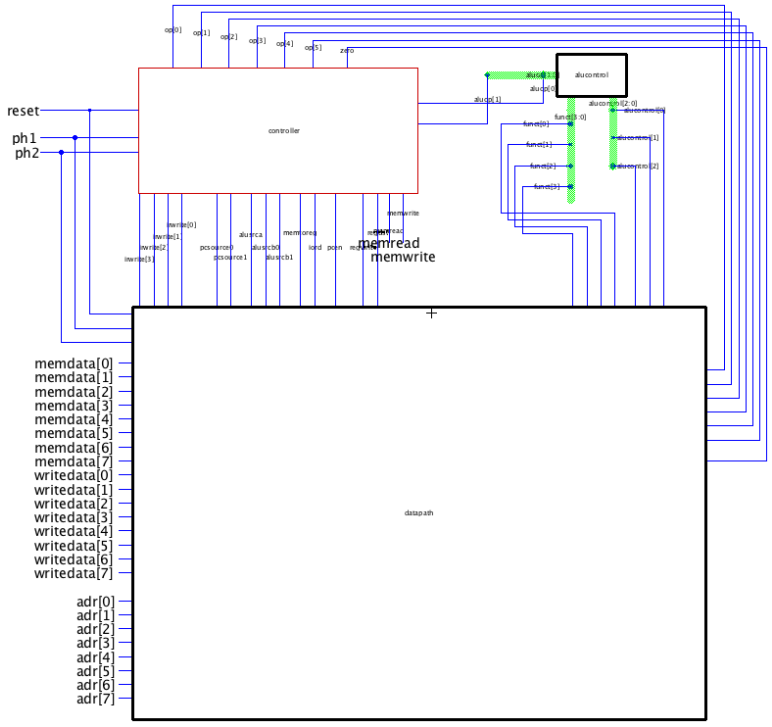
See Section X on how to name and organize the files that you will need to submit for this lab.

IV Collaboration

You must complete this lab independently; you are not allowed to work in pairs or a group. However, you are welcome to discuss the material with your colleagues.

V Prerequisites

You must have completed Labs #1-4 before proceeding further. If not, then it your responsibility to do that first. Otherwise, you will be totally lost in this lab.



Inputs	Outputs
ph1	adr[7:0]
ph2	writedata[7:0]
reset	memread
memdata[7:0]	memwrite

Figure 1: Top-level schematic of the MIPS processor.

VI Provided Files

All the files for this lab are provided in `lab5.zip`. Unpack this archive and you will see the following directory structure:

```
lab4/
├─README                          readme
├─lab5-xx.jelib                  Electric library
├─iopads-ami05.jelib             Pads for AMI 0.5 μm
├─mips8-sch.arr                  Pad arrangement file for schematic
└─mips8-lay.arr                  Pad arrangement file for layout
  sim/                           IRSIM test vectors
  └─mips.cmd
```

The `lab5-xx.jelib` file provides all the cells for this lab. However, the cells `std_nand3`, `fulladder`, `alu`, `alucontrol`, `bitslice`, `zipper`, `controller`, and `datapath` are incomplete. Replace them with those that you have designed in the previous labs. The `iopads-ami05.jelib` file contains the I/O pads. The `*.arr` are the pad arrangement files. To test the complete MIPS processor, a sample (yet incomplete) test program is given in `mips.cmd`.

VII Assembling the MIPS Processor

Follow the directions in each subsection to assemble the processor in its entirety.

VII.A Top-level Schematic Simulation

The top-level schematic of the MIPS processor is present in `mips8{sch}`. It is also shown in Fig. 1, and includes the datapath, alucontrol, and controller icons. The system's exported inputs and outputs are also shown. All

```
; Sample test program to test the 8-bit MIPS micorprocessor

        ; Assembly code        Effect                      Machine code
main:   lb $2, 68($0)         ; initialize $2 = 5         80020044
        lb $7, 64($0)         ; initialize $7 = 3         80070040
        lb $3, 69($7)         ; initialize $3 = 12        80e30045
        or $4, $7, $2         ; $4 <= (3 or 5 = 7)        00e22025
        and $5, $3, $4        ; $5 <= (12 and 7 = 4)      00642824
        add $5, $5, $4        ; $5 <= (4 + 7 = 11)        00a42820
        beq $5, $7, end       ; shouldn't be taken        10a70008
        slt $6, $3, $4        ; $6 <= (12 < 7 = 0)        0064302a
        beq $6, $0, around    ; should be taken           10c00001
        lb $5, 0($0)          ; shouldn't happen          80050000
around:
        slt $6, $7, $2        ; $6 <= (3 < 5 = 1)         00e2302a
        add $7, $6, $5        ; $7 <= (1 + 11 = 12)       00c53820
        sub $7, $7, $2        ; $7 <= (12 - 5 = 7)        00e23822
        j end                 ; should be taken           0800000f
        lb $7, 0($0)          ; shouldn't happen          80070000
end:
        sb $7, 0($2)          ; write adr 5 <= 7          a0470000
```

Figure 2: Sample test program for the MIPS processor.

the interconnections have been made so there is nothing that you have to add to the schematic. Notice that the internal wires and icons are also named to simplify debugging. Once you have copied over the necessary cells from the previous labs, run DRC on this schematic. Do not proceed further until all the problems have been fixed and hierarchical DRC passes.

To demonstrate basic functionality of your processor, you will simulate it with a test program shown in Fig. 2. Since this is a multi-cycle processor, each instruction will require several steps. The `mips.cmd` file has been provided to you with the commands to simulate most of the instructions. However, instructions 3 (`lb $3, 69($7)`) and 4 (`or $4, $7, $2`) are missing. Add them to the command file with the appropriate assertions. Use IRSIM to simulate the schematic. If you receive any `Pending events` warnings, these can be ignored. However, there should not be any assertion violations. Fix any errors that you encounter in the design.

VII.B Top-level Layout

Create the layout of the processor in `mips8{lay}`. In this top-level layout, place the datapath, alucontrol, and controller cells so that they will be easy to connect. Wire together the modules. Don't forget to connect the power and ground lines with fat (width = 20) wires and arrays of vias to handle the higher levels of current that may flow! You will avoid creating problems and headaches for yourself if you systematically reserve `metal1/metal3` and `metal2` for horizontal and vertical lines, respectively. You may wish to consider placing a large number of long horizontal wires between the datapath and the controller/alucontrol, and then drop vertical lines in a systematic fashion to connect the signals between the cells together.

Export all inputs, outputs, and power and ground lines. Label the internal signals for ease of debugging. Make sure all your labels and exports agree with the schematic (`mips8{sch}`). Verify that the entire layout passes DRC, ERC, NCC, and IRSIM simulation. Fix any problems that might arise. Choose Preferences→Tools→DRC and click on Clear valid DRC dates to reset the DRC timestamps. Then, rerun DRC, ERC, and NCC on the top-level layout to ensure everything is passing.

VII.C Pad Frame 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 μm^2; these pads are large enough to be attached to the package during

manufacturing with thin gold bonding wires. Each pad contains large transistors to drive the relatively enormous capacitances of the external environment.

Electric provides a handy pad frame generator that automatically assembles a pad frame for you from your library. To use the pad frame generator, you need your library, a I/O pad library, and a pad arrangement file. The I/O pad library is provided in `iopads-ami05.jelib`. Two pad arrangements are provided in `mips8-sch.arr` and `mips8-lay.arr`.

Open and browse `mips8-sch.arr`. The `celllibrary` defines the library containing the I/O pads. The `cell` and `views` defines the name of the cell and facet to generate. The `core` defines the top-level cell to be used as the chip's core (in this case, its `mips8`). The `place` commands list the I/O pads, its type, and the connection to the core. The `export` command is used to export the signal name. Note that there are 10 I/O pads on each side of the chip with at least two pads each dedicated to supply power and ground to the pads and core.

Choose Tools→Generation→Pad Frame Generator. Select the `mips8-sch.arr` file. This will create a new facet named `chip{sch}` with *generic unrouted arcs* connecting the pads to the `mips8` core. Inspect the pad frame to ensure it is reasonable. Each pad has a V_{DD} and GND export on both edges that must be connected to the adjacent pad. These connections should have already been made for you by the pad frame generator. If they are not present, make the connections manually. Run DRC to make sure there are no errors.

Generating `chip{sch}` will open `iopads-ami05.jelib` and leave you with two open libraries. You can change the "current library" by selecting a library, right clicking, and choosing Make This the Current Library. Make sure `lab5-xx.jelib` is the "current library".

Repeat the above process for the layout using `mips8-lay.arr`. This will generate `chip{lay}`. The core might not be positioned at the center of the chip. If so, select the core and drag it until it is approximately in the center. You will note that generic unrouted arcs are connected between the pads and core. In a real design, you will need to delete the generic unrouted arcs in the layout and replace them with real metal lines. This is called *global routing* and is sufficiently tedious that it is usually done with an automatic routing tool. Unfortunately, we do not have such a tool available so you will have to do it manually.

Delete the generic unrouted arcs and systematically connect the core outputs with the pads. Use `metal3` and `metal2` wires for horizontal and vertical routing, respectively. Once the routing is complete, ensure that your layout passes DRC, ERC, and NCC. Fix any problems that might arise.

Important: When you run NCC on `chip{lay}`, Electric will complain about "implied" V_{DD}/GND signals present in the schematic that are not present in the layout. You can ignore these errors. However, make sure you fix any other errors.

Do a thorough final check of the top-level chip layout using the following steps and fix any errors:

1. Choose File→Check Libraries→Check: Makes sure library is stable before verification proceeds.
2. Choose File→Preferences→Tools→DRC and click on Clear valid DRC dates.
3. Run DRC, ERC, and NCC (ignore implied V_{DD}/GND errors) on `chip{lay}`.
4. Simulate `chip{lay}` using IRSIM and `mips.cmd` (you will have to change the signal names of the buses to match those on the chip). The command file can also be used to test the chip after it is manufactured.

VII.D Tapeout

The final step in designing a chip is creating a file containing the geometry needed by the semiconductor vendor (*e.g.*, MOSIS, TSMC, *etc*) to manufacture masks. In the older days, these files were written to magnetic tape, and the process is still known as *tapeout*. The two popular output formats are Caltech Interchange Format (CIF) and Graphic Data System (GDS); we will use GDSII. Choose File→Export→GDS II (Stream). Save your design in `mips8-chip.gds`. This file can be sent to the vendor to fabricate the chip.

GDSII is a binary stream format. Thus, it is not possible to inspect the format through a text editor. However, you can view the file in a GDSII viewer. A free publicly available viewer is OwlVision which also allows you to

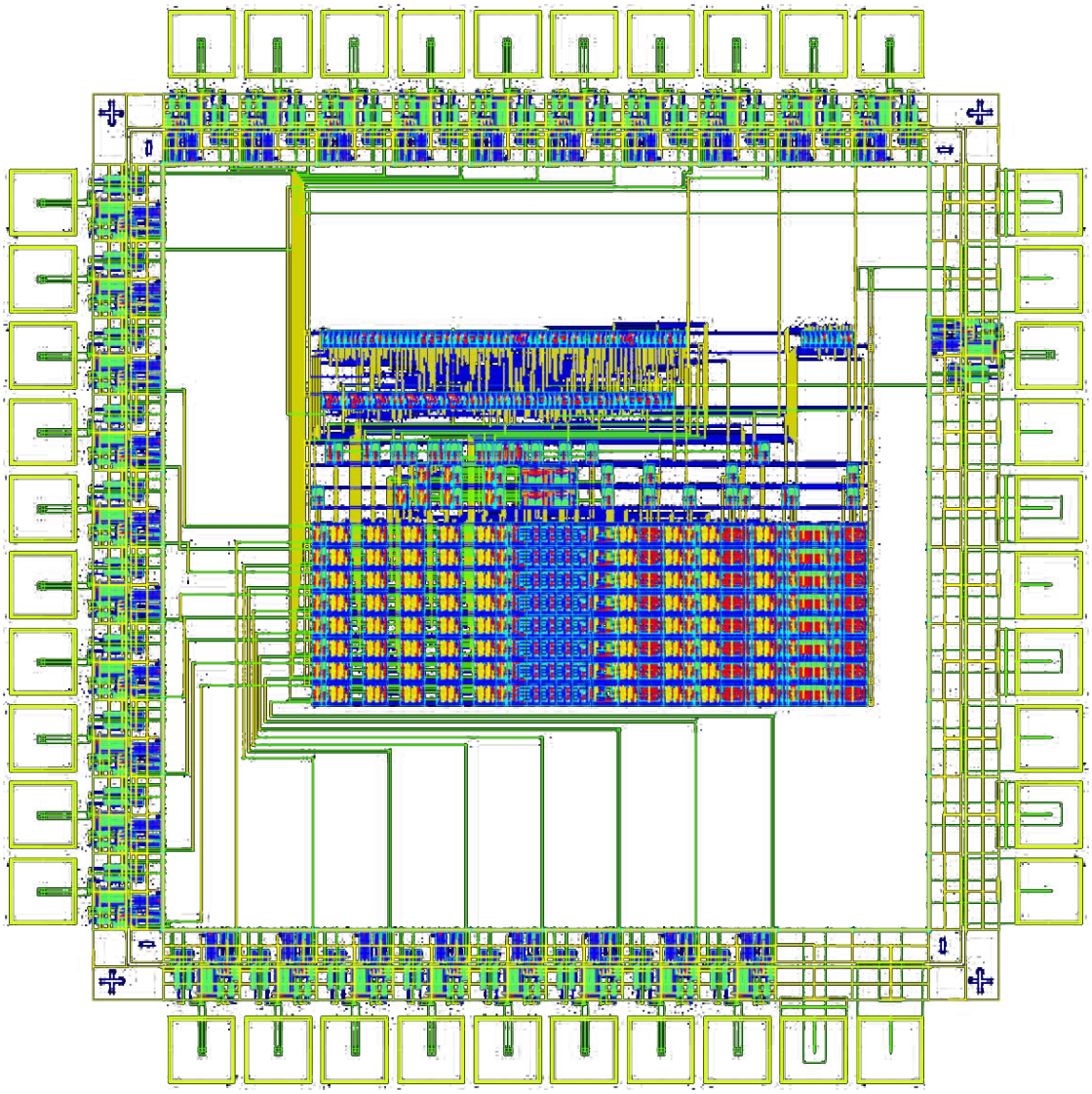


Figure 3: Screenshot of our 8-bit MIPS processor.

save a screenshot of the chip. Fig. 3 shows a screenshot of the 8-bit MIPS processor (your chip layout should be similar to this)!

VIII Technical Report

A technical report (not to exceed 10 pages) is to be written that details everything you have done in this lab. You should present the design of the entire chip. You should present the appropriate schematics and layouts. Furthermore, you should show the results of any simulations, and whether the layout passed DRC, ERC, and NCC or not. You should also include an OwlVision screenshot of your chip. Elaborate on any difficulties faced in this lab and the employed workarounds. Summarize what you have learned from doing this lab.

The technical report must be of the highest standards. Otherwise, it runs a high risk of being rejected which will impact your lab grade. You can consult some technical publications to see how to write a good technical report.

It must be written using the LaTeX template that was provided earlier. The template can be downloaded at http://pandim.ece.villanova.edu.

IX Parting Words

Congratulations on finishing this lab! You have successfully designed, assembled, and tested your own microprocessor. You are now familiar with the major aspects of *custom* VLSI design:

- Datapath and standard cell design
- Datapath design (sliceplan, floorplan) and assembly
- Standard cell synthesis and place & route
- Top-level system assembly
- Pad frame generation and routing
- Switch-level simulation (IRSIM) and logic debug
- Verification: DRC, ERC, and NCC

You will put these skills to use as you proceed with your final project!

X What to Submit

For this lab, you must submit the following files:

1. The Electric library. Name it `lab5-xx.jelib` where `xx` are your initials.
2. The IRSIM test vector file for the entire MIPS that includes instructions 3 and 4. Name it `mips.cmd`.
3. A GDSII file containing your design. Name it `mips8-chip-xx.gds` where `xx` are your initials.
4. The LaTeX PDF file which contains the technical report. Name it `report-xx.pdf` where `xx` are your initials.

Take all the files and archive (zip) them into a folder. Name the folder `lab5-[first name]-[last name].zip`. See Section III on how to submit the archive. Failure to follow these instructions will result in a **zero** for the lab. No ifs, buts, *etc.*

Important: The Electric library must contain the schematics, icons, and layouts of the cells used in constructing the chip. All schematics must pass DRC and IRSIM simulation. The layouts must pass DRC, ERC, NCC, IRSIM simulation, and be drawn as per specification. Icons must be of the correct size.

XI Errors

I usually write precise tutorials and bug-free code. However, I am human (do not be surprised) and do make mistakes. In addition, CAD tools get updated frequently and the interface might change, rendering parts of the writeup ineffective. If you find any mistakes or inconsistencies while doing this lab, please bring it to my attention **immediately**. You will earn extra points if what you report is indeed a bug.