ECE 3450: Digital Electronics
Fall 2009, Lab #3

Dr. Pallav Gupta
pallav.gupta@villanova.edu

I Objectives

The goals of this lab are the following:

1. Use the cells designed in earlier labs to layout the ALU of a simplified MIPS processor.
2. Design the schematic of an n-bit datapath by designing a horizontal bitslice.
3. Layout the zipper to drive the control signals onto the datapath.
4. Layout the 8-bit datapath of the simplified MIPS processor.
5. Become proficient in editing relatively large-scale designs.

II Courses Forum

The forum is located athttp://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:	`lab3-[first name]-[last name].zip`, *e.g.*, `lab3-john-doe.zip`
Location:	`ECE 3450/LAB3`
Title:	`Lab 3`
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 #1-2 before proceeding further. If not, then it your responsibility to do that first. Otherwise, you will be totally lost in this lab.

VI Provided Files

All the files for this lab are provided in `lab3.zip`. Unpack this archive and you will see the following directory structure:

```
lab3/
├─README                                  readme
├─lab3-xx.jelib                          Electric library
└─sim/                                   IRSIM test vectors
  ├ alu.cmd
  └ datapath.cmd
```

The Electric file provides a small library of complex cells that will be necessary to complete this lab. Browse through to understand them. The library does not provide the fulladder. You should use the cell that you designed in the previous lab. Copy over the necessary cells into this library.

You should focus your attention on the following four cells:

1. Arithmetic Logic Unit (ALU) - `alu{sch}`, `alu{ic}`, `alu{lay}`
2. Bitslice - `bitslice{sch}`, `bitslice{ic}`, `bitslice{lay}`
3. 8-bit datapath - `datapath{sch}`, `datapath{ic}`, `datapath{lay}`
4. Zipper - `zipper{sch}`, `zipper{ic}`, `zipper{lay}`

The `alu.cmd / datapath.cmd` contains IRSIM test vectors to test the ALU and datapath, respectively. Note that a lot of the design has already been finished for you. To prevent problems integrating your additions, it is vital that you do not edit/change what has been provided to you.

VII Block Diagram of the Simplified MIPS Processor

The block diagram of the simplified 8-bit MIPS processor is shown in Fig. 1. It consists of a controller, datapath, and alucontrol. The controller comprises the control finite-state-machine (FSM) to generate the control signals. The alucontrol generates the control signals to drive the ALU. The datapath contains the remainder of the chip, organized as 8 identical bitslices. Study the block diagram until you are familiar with the organization and the various signals. You may also want to consult Chapter 1 of the textbook.

The processor has no on-chip memory; instead, it provides an interface to external SRAM. The interface consists of three 8-bit buses. Two are outputs containing the address (`adr`) and data (`writedata`) to write to the memory. The third is an input carrying the data (`memdata`) read from the external memory.

VII.A Designing the ALU

The design of a simple 1-bit ALU sans overflow detection of the simplified MIPS processor is shown in Fig. 2. The ALU implements five instructions: `ADD`, `SUB`, `AND`, `OR`, and `SLT`. Refer to Chapter 1 of the textbook to see the operations of these instructions. Study the schematic shown in Fig. 2 until you understand its functionality.

The schematic and icon of the ALU are already provided in `alu{sch}` and `alu{ic}`, respectively. Verify that the schematic passes DRC. Simulate the schematic using the IRSIM test vectors provided in `alu.cmd`, and observe

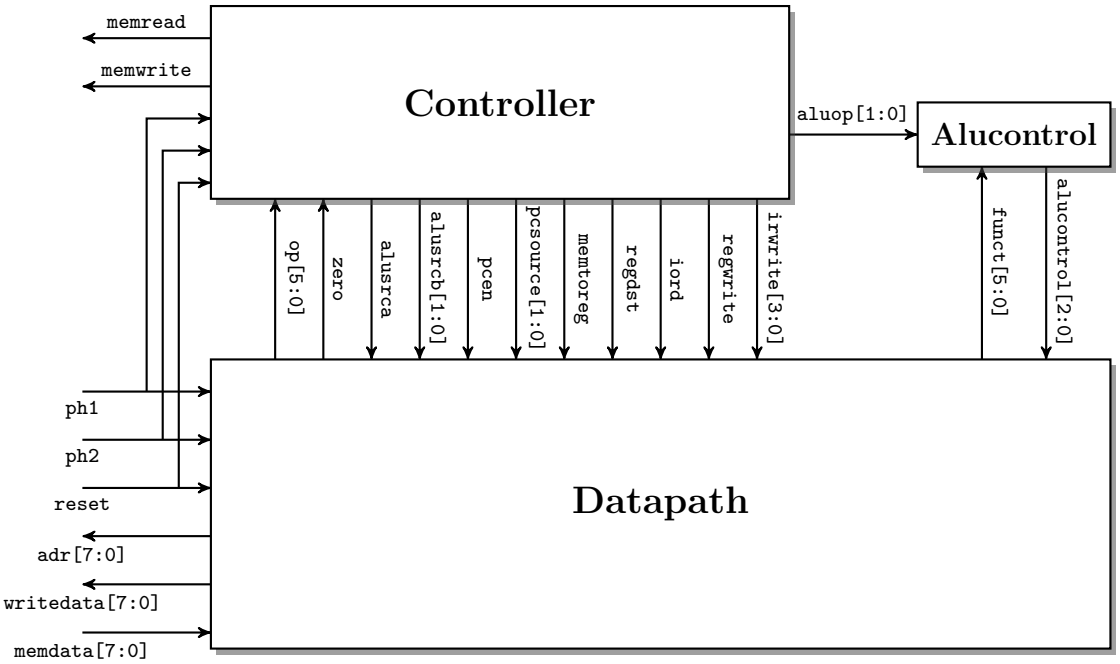


Figure 1: Block diagram of the simplified 8-bit MIPS processor.

the waveforms. Make sure you understand the behavior of the ALU (*i.e.*, instruction execution). You should be able to identify which instruction is being executed at a given time.

Your job is to layout the ALU of Fig. 2 in `alu{lay}`. Place the various gates in a horizontal line and route the connections to match the schematic. Use `metal1` and `metal2` wires for horizontal and vertical routing, respectively. You can also use `metal3` wires for horizontal routing (if there is congestion) but keep in mind that they run on a 10λ pitch (5λ width, 5λ spacing). Export all the inputs, outputs, control signals, and power and ground lines. Refer to `alu{sch}` to see all the exports.

Verify the layout by running DRC, ERC, and NCC. Simulate your ALU layout using `alu.cmd` in IRSIM to ensure it functions correctly. Check that all instructions are working properly. Periodically check and repair your library to catch other problems. Remember to save your work at regular intervals. Electric is known to crash.

Here are a few hints and tips to keep in mind while drawing the layout:

1. Start from the left of the schematic and work your way to the right, placing the cells in order.
2. Draw a rough sketch before starting the actual layout.
3. The cells must be placed horizontally with a 90λ center-to-center spacing.
4. Use `metal1` (8λ pitch), `metal3` (10λ pitch) for horizontal routing and `metal2` (8λ pitch) for vertical routing.
5. Strive to keep everything aligned on a 1λ grid.
6. Use Cell → Cross – Library Copy to copy necessary cells from your other libraries.
7. Use Cell → Expand Cell Instances → All the Way to view the contents of a cell; choose Unexpand Cell Instances to zoom out of a cell.
8. Use `ctrl-click` to cycle through the various layers when you have lines drawn on top of facets. Review the Electric manual to pick up the finer nuances of selecting from a stack of many different objects.
9. Place large pure-layer nodes for the N-well and P-well to avoid difficulties with gaps between wells.

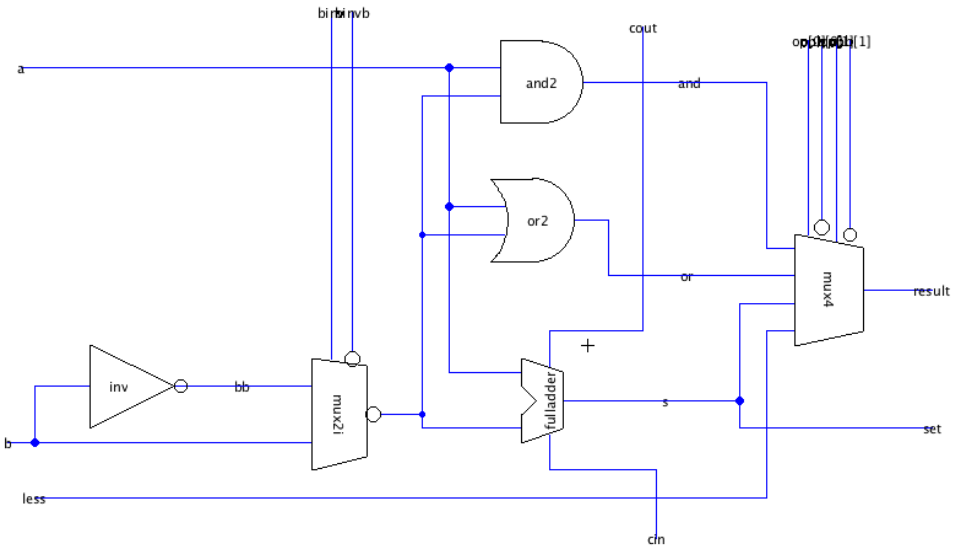


Figure 2: Schematic of a 1-bit ALU.

10. Use `metal1`, `metal2`, or `metal3` pins from the palette to give yourself a destination to connect to when Electric gets confused about snapping a connection to an undesired destination. Select two nodes, then right click on a blank space to connect the nodes.
11. Use the red boxes in the palette window to choose which layer will be drawn in the event of ambiguity. For example, when connecting two vias, select the appropriate metal line to indicate the connection.
12. Use Export → {List|Show} Exports to get the name(s) and type(s) of export(s) on a network.
13. Export all signals on `Metal1-Metal2` contacts.
14. Expect to spend some time playing around with Electric to become familiar with how it makes connections when there are many layers of stuff (*i.e.*, stack).

VII.B Designing the Datapath

Next, you will design the datapath of the simplified MIPS processor. Since this is an 8-bit processor, the datapath is 8 bits wide. However, because of a very regular structure, we can construct an 8-bit datapath from 8 identical horizontal bitslices. Data signals travel horizontally along the bitslice. Control signals run vertically to all 8 bitslices of the datapath. A small amount of logic is required to generate the control signals. For example, a multiplexer in the datapath requires true and complementary select signals. Rather than provide a local inverter in each of the 8 bitslices, the inverter could be placed on top of the datapath in a *zipper* to drive the complementary signal to all the bitslices.

The slice plan of the datapath is shown in Fig. 3. The diagram illustrates the ordering of cells in the bitslice and the allocation of wiring tracks. Wires at the left and right side of a cell are inputs and outputs, respectively. Dots indicate that a wire passes over a cell and is also used in that cell. Each cell is annotated with its type and approximate width (could change depending upon the layout of the cell). For example, the program counter (`pc`) is an output of the PC flop and is also used as an input to the `srcA` and address (`adrmux`) multiplexers. Study the slice plan in detail until you understand it.

Open up `bitslice{sch}` and compare it to Fig. 3. On the left side of the bitslice is the address multiplexer (`adrmux`) selecting the address for external memory. The 32-bit instruction is stored in four 8-bit instruction registers (it takes four cycles to fetch an instruction from memory), so four flip-flops named `ir3`, `ir2`, `ir1`, and `ir0` are required in each bitslice. The memory data register is named `mdr`. Next comes the interface to the

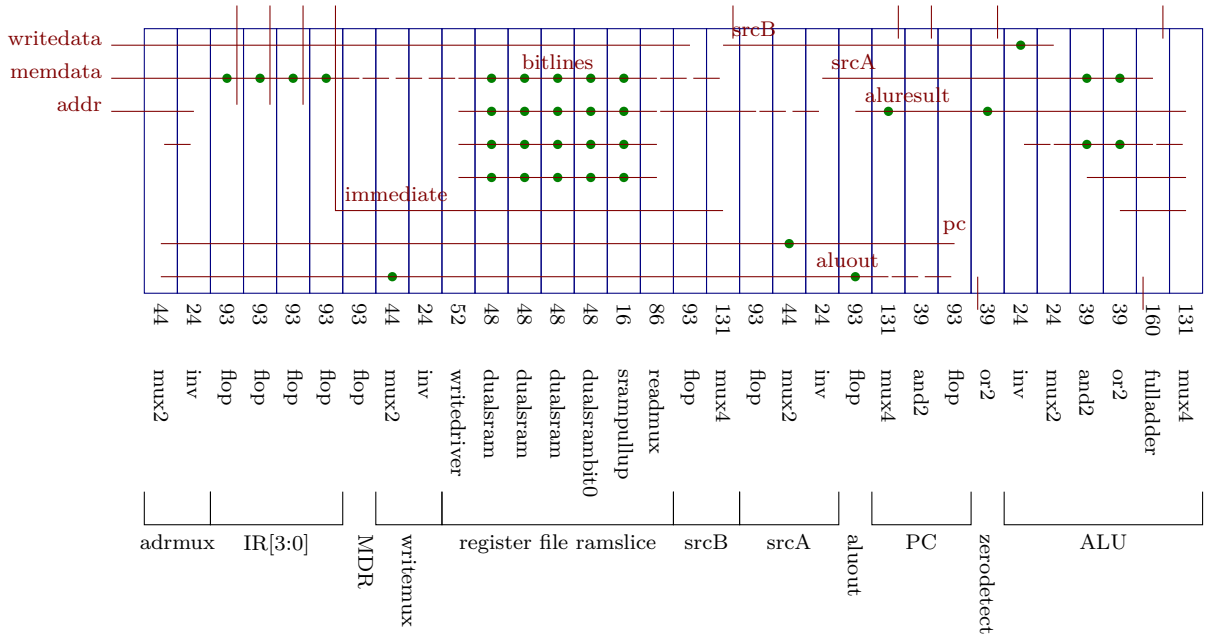


Figure 3: Datapath slice plan.

register file. This consists of the write data multiplexer, `wdmux`, the 8-word register file itself (`ramslice`), and the source registers (`areg`, `breg`). Interdigitated with the registers are `src1mux` and `src2mux` multiplexers that chose the operands for the ALU and the `aluoutreg` register. The program counter logic consists of the multiplexer to choose the next value of the program counter, an AND gate to reset the program counter to 0 on startup (reset), and the program counter flop (`pcreg`) itself. At the very right end of the bitslice is the ALU.

Your job is to design the bitslice schematic, `bitslice{sch}`. To ease your task, all the necessary cells have been placed in their appropriate destinations. All the relevant signals have been exported. In addition, the various clock signals to the flip-flops and the multiplexer select signals have already been connected. You are required to finish the design by making the remaining connections. Table 1 indicates the signals that need to be connected to the various inputs of the various multiplexers. Make sure you do not violate Table 1. The other connections that are required can be deduced from the slice plan in Fig. 3. Once the wiring is complete, verify the schematic by running DRC.

After designing the bitslice schematic, finish the layout in `bitslice{lay}`. You only need to add and connect the ALU that you designed earlier to the remainder of the pre-designed bitslice. Do **not** change or move anything in the existing layout (otherwise you will have many problems assembling the datapath). Verify that your layout passes DRC, ERC, and NCC. Ensuring that the layout passes NCC will also validate your schematic. Fig. 5a shows a screenshot of the entire bitslice. Finally, verify the datapath by simulating it in IRSIM using `datapath.cmd`.

VII.C Designing the Zipper

The zipper takes the control signals generated by the controller and drives them onto the datapath. If necessary, it also produces the complement of the control signals. The design of the zipper is shown in Fig. 4. It consists of some multiplexers which feed into decoders that generate the signals to select the appropriate registers (source operands) in the register file (`ramslice`). It also contains strong buffers (4x) that generate the complement of a control signal and drive both signals onto the datapath. Study Fig. 4 until you understand it.

The schematic and icon of the zipper are already provided in `zipper{sch}` and `zipper{ic}`, respectively. Verify that the schematic passes DRC. Your job is to layout the zipper in `zipper{lay}`. To ease your task a little,

Table 1: Multiplexer Input Connections

Mux name	Input	Input Signal
adrmux	D_0	$Q_{\texttt{pcreg}}$
	D_1	$Q_{\texttt{aluoutreg}}$
writemux	D_0	$Q_{\texttt{aluoutreg}}$
	D_1	$Q_{\texttt{mdr}}$
src2mux	D_0	$Q_{\texttt{breg}}$
	D_1	four
	D_2	instr0 ($Q_{\texttt{ir0}}$)
	D_3	instrshift
src1mux	D_0	$Q_{\texttt{pcreg}}$
	D_1	$Q_{\texttt{areg}}$
mux4	D_0	$\texttt{result}_{\texttt{alu}}$
	D_1	$Q_{\texttt{aluoutreg}}$
	D_2	instrshift
	D_3	GND

the cells have been placed in their exact locations. Do **not** move the cells (otherwise, you will encounter many problems in assembling the datapath). Finish the layout by routing the connections to match the schematic. Use `metal1` and `metal3` for horizontal and `metal2` for vertical routing. Export all the signals on `Metal1-Metal2` contacts. Also export power and ground lines. See `zipper{sch}` for the complete list of exports. Verify the layout by running DRC, ERC, and NCC. Fig. 5b shows a screenshot of the zipper.

VII.D Assembling the Datapath

Now that the bitslice and zipper have been completely designed, you can assemble the datapath in `datapath{sch}`. The datapath consists of eight bitslices and the zipper for driving the control signals. Study the schematic to understand the organization. Again, all the cells have been placed on the schematic. The relevant signals have been exported. Some of the non-trivial connections have already been completed. You are required to finish the design by making the remaining connections. Essentially, all bitslices are fed the same control signals from the zipper. Thus, connect the remaining signals vertically in the bitslices. Verify the schematic by running DRC.

Finally, finish the layout of the datapath in `datapath{lay}`. To ease your task a little, the bitslices have been placed, and the power and ground lines have been routed. The top bitslice has been connected with the zipper. Observe the wiring tracks between the top bitslice and zipper. Zoom in to analyze the existing connection and understand them well. All the appropriate signals have been exported. If there are any that are missing, just add them. Connect the other signals on the remaining bitslices appropriately. You should use `metal2` for vertical routing, and `metal1` and `metal3` for horizontal routing. Verify the datapath by running DRC, ERC, and NCC. Fig. 6 shows a screenshot of the entire datapath including the zipper. Notice the regularity in structure and the denseness. This is because we started with a detailed and well-crafted slice plan which guided our design. Had we not done this, the layout could have been inefficient (more area, irregular, *etc*) or taken a lot more time to assemble.

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 ALU, bitslice, datapath, and zipper. 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. Elaborate on any difficulties faced in this lab and the employed workarounds. Summarize what you have learned from doing this lab.

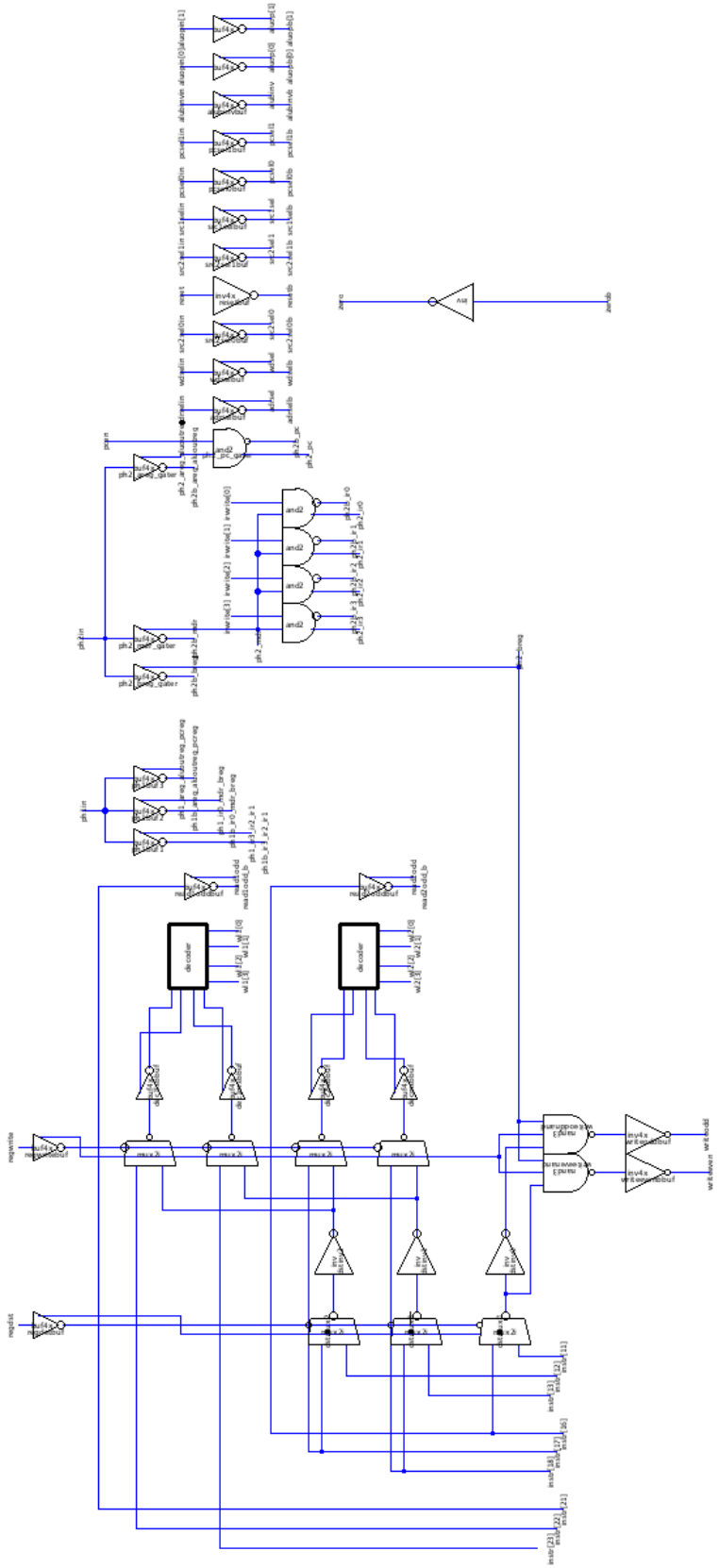


Figure 4: Schematic of the zipper.

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! Hopefully, you now have some experience in handling relatively large-scale designs and know the basics of ALUs, slice plans, bitslices and datapaths, and zippers. Remember that good layouts come from good slice plans/floorplans! Thus, it is critical to spend time creating one before performing any actual layout.

X What to Submit

For this lab, you must submit the following files:

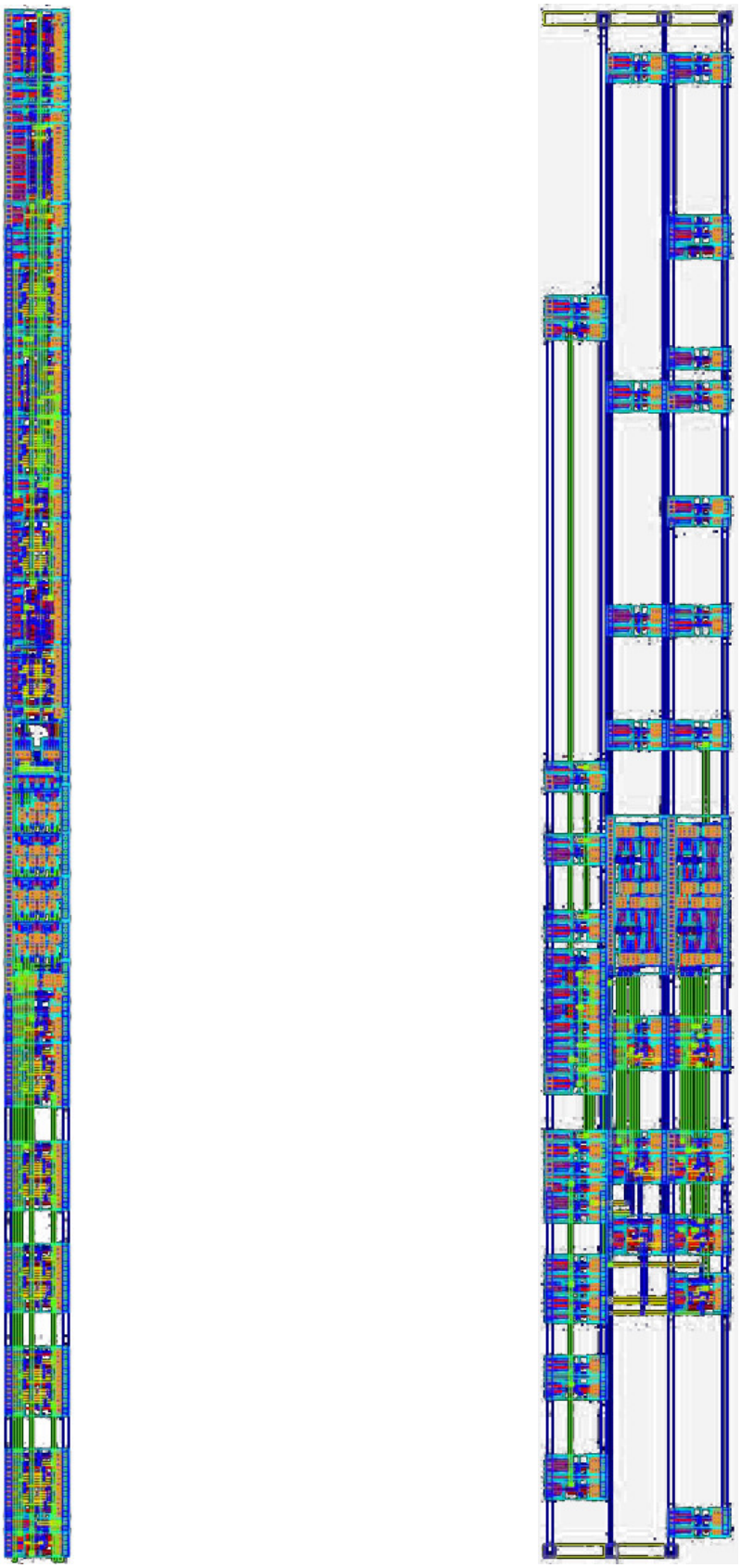
1. The Electric library. Name it `lab3-xx.jelib` where `xx` are your initials.
2. The LaTeX PDF file which contains the technical report. Name it `report-xx.pdf` where `xx` are your initials.

Take both the files and archive (zip) them into a folder. Name the folder `lab3-[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 ALU, bitslice, datapath, and zipper. All schematics must pass DRC and IRSIM simulation. The layouts must pass DRC, ERC, NCC, IRSIM simulation, and be drawn as per specification (outlined above). Icons must be of the correct size. Furthermore, if you are using any cells from previous labs, their complete design needs to be present in the library.

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.



(a) Bitslice. (b) Zipper.

Figure 5: Screenshots of the bitslice and zipper.

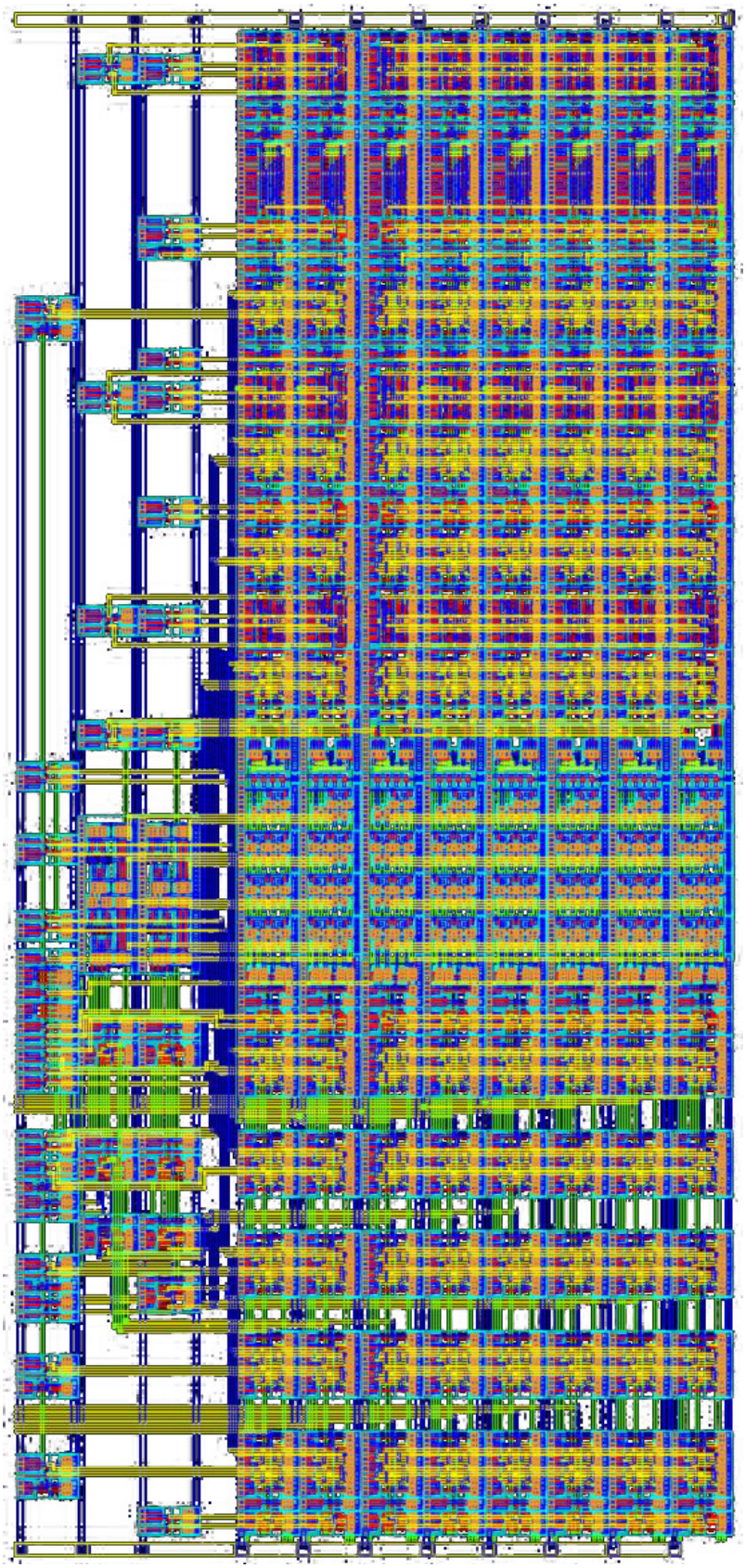


Figure 6: Screenshot of the datapath of our 8-bit MIPS processor.