Harris

Introduction to CMOS VLSI Design (E158)

Lab 3: Datapath and Zipper Assembly

An *n*-bit *datapath* consists of *n* identical horizontal *bitslices*[1]. Data signals travel horizontally along the bitslice. Control signals run vertically to all *n* bits of the datapath. Often, 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 *n* bitslices, the inverter could be placed on top of the datapath in a *zipper* to drive the complementary signal to all the bitslices.

Most of the processor datapath has been provided for you, but the ALU is missing from each bitslice. In this lab, you will add your ALU to the bitslice and wire it in to the system. You will then place necessary inverters in the zipper and connect them to the datapath.

Although the total amount of layout to do in this lab is modest, you will find Electric has a learning curve before you become proficient editing large designs like the MIPS datapath.

1. Changes for 2001

Before you begin, please patch up some junk that was incorrectly left in the e158 library. Open your lab2_xx.elib. Look at the datapath{lay} facet. In the far right side are a bunch of parallel metal2 lines with vias at the right ends. Near the upper right corner of these lines sits a lonely via. Delete the lines, the vias they attach to, and the lonely via. Then look at zipper{lay}. At the right end are three buf4x circuits connected to vertical metal1 signals *alubinvin*, *aluopin*[0], and *aluopin*[1]. These were supposed to be missing from your lab, but were left in the zipper inadvertently. When you get to the section editing the zipper where you are supposed to add these buffers, you can instead just move them around to the appropriate positions.

2. Bitslice Assembly

Copy your lab2_xx.elib to lab3_xx.elib and open the new library. Look at the datapath bitslice schematic dpbitslice{sch}. It is complete. Zoom in so that you can read the

[1] In general, the bits of the datapath might vary slightly. For example, a carry lookahead adder requires different logic in different bits. And of course, the horizontal/vertical distinction is arbitrary and a datapath could be rotated.

labels on each icon. Match the hardware in the schematic to the MIPS datapath in Figure 1 of Lab 1.

The processor has no on-chip memory; instead, it provides an interface to external SRAM. The interface consists of 3 8-bit busses. Two are outputs containing the address and data to write to the memory. The third is an input carrying the data read from the external memory.

On the left side of the datapath is the address multiplexer (adrmux) selecting the address for the external memory. The 32-bit instruction is stored in four 8-bit instruction registers, so four flip-flops named ir3…ir0 are required in each bitslice. The memory data register is named mdr. Next comes the interface to the register file. This consists of the write data multiplexer wdmux, the 8-word register file itself (drawn below the datapath to keep the entire schematic on one page), and the A and B registers. Interdigitated with the A and B registers are the Source 1 and Source 2 multiplexers choosing the operands for the ALU and the ALUOut 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, and the program counter flop itself. At the very right end of the bitslice is the ALU.

The dpbitslice{lay} is complete except for the ALU that you designed in Lab 2. Look at the layout and relate the cells in the layout to the cells in the schematic. Double-click on each cell to view its name. At the very left end, you again find the adrmux. Double-click on the other cells and verify that their names match the schematic. VDD and GND run along the top and bottom of the datapath, respectively. Notice how metal2 *bitlines* on an 8λ pitch are used to connect the cells together. For example, *adr*, *memdata*, and *writedata* exit the bitslice to the left. These signals will ultimately connect to the external memory. Notice how *memdata* runs over the top of the adrmux and to the inputs of the four instruction registers.

Now that you are familiar with the layout, add your ALU layout to the right end of the dpbitslice layout. Zero-detect logic will eventually go between the ALU and the remainder of the dpbitslice, so place your ALU at least 60λ right of the existing circuitry. Connect VDD and GND to the adjacent flip-flop. Note that these ports are on the left end of the flip-flop. As shown in the schematic, make three connections to the rest of the datapath: inputs *A* and *B* come from *src1* and *src2*, and output *result* goes to the *aluresult* signal. In the datapath layout style, these connections should occur using mostly horizontal metal2 lines. The lines must run over the top of the cells, not above VDD or below GND.

Here are a few hints while drawing the layout:

- Use Facet • Expand Facet Instances to view the contents of a cell.
- Unexpand the facet instances to make it easier to see exports on facets
- Use Windows • Alignment Options to set a 0.5 lambda grid when necessary.

- Ctrl-click to cycle through various different layers when you have lines drawn on top of facets.
- The layout already has some large pure-layer nodes for the N and P wells. You may need to double-click on the wells and resize them to fit your design.
- Use Info • List Exports on Network to get the name of the export on a selected network.
- Use metal1 or metal2 pins from the palette to give yourself a destination to connect to when Electric gets confused about snapping a connection to an undesired destination.
- 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 either the blue or purple lines to indicate metal1 or metal2 for the connection.
- Expect to spend some time playing around with Electric to come to understand how it makes connections when there are many layers of stuff.

Add exports to the signals emerging from the top and bottom of the ALU. These include inputs *alubinv*, *alubinvb*, *aluop*[0], *aluopb*[0], *aluop*[1], *aluopb*[1], *set*, and *cin*, and outputs *less* and *cout*.

When your changes are complete, use DRC, ERC, and NCC to verify your layout.

3. Zipper Modifications

Recall that the ALU requires true and complementary versions of *alubinv*, *aluop*[0], and *aluop*[1]. The controller that you will design in Lab 4 only produces the true version of each signal. Therefore, we must locally invert the signal. Moreover, each signal must drive eight bits of the datapath. For good performance, we would like to drive these bits with something larger than a minimum-sized inverter. Thus, in the zipper we provide buf4x drivers for each control signal. These accept the inputs from the controller and use inverters with 4 times the usual transistor widths to drive true and complementary control signals across the datapath.

Look at the datapath{lay}. You will see eight rows of the dpbitslice that you have just completed. Above the bit slices is the zipper that generates the control signals for all the bits of the datapath. Look at the zipper{lay}. The center portion contains the decoder for the register file. The right portion contains a bunch of buf4x cells to drive control signals across the datapath. It is missing three buf4x drivers for *alubinv*, *aluop*[0], and *aluop*[1], as shown in the zipper schematics. Add these three buffers. Align the buffers above the datapath so the outputs of the zipper are properly located to drive the multiplexer control lines that you exported from the dpbitslice. Be sure the port names match the zipper schematics.

Run DRC, ERC, and NCC to verify the zipper.

4. Datapath Assembly and Verification

Your final task is to assemble and verify the datapath. Vertical metal1 control lines tie the bitslices together. Most are already complete, but you must add the ones for the ALU.

First add the six lines for *alubinv*, *aluop*[0], *aluop*[1], and their complements. These lines should connect the zipper outputs to each of the eight bitslices.

Next, connect the carry chain. As shown in datapath{sch}, the carry into the least significant bit should be tied to binv so it is 1 for subtractions and 0 for adds. This bit is the bottom bitslice in the datapath. You will have to run a metal2 line carrying ground all the way from the left end of the bottom bitslice to reach this carry input. For each of the remaining bits, *cin* should be connected to *cout* of the bit below. This is easiest if the carry in and carry out signals are located in the same vertical column, as you were instructed to do in Lab 2.

Finally, connect the *Less* signal used for the set on less than (`slt`) instruction. Recall that `slt` produces a 1 if input A is less than input B and a 0 otherwise. This can be accomplished by computing A-B. If the result is negative, indicated by a 1 in the most significant bit of the subtraction, set the output to 1. Otherwise, set it to 0. In other words, the least significant bit of the output should equal the most significant bit of the subtraction result. All other bits should be 0. As shown in the datapath schematic and in Figure 4.18 of the text, the least significant *Less* input connects to the *set* output of the most significant bit. Remember that the least significant bit is the bottom row of the datapath and the most significant bit is the top row. All other *Less* inputs should be tied to ground.

Run DRC, ERC, and NCC to verify the datapath. For a design this large, tracking down errors is very difficult. Therefore, be sure you know what you are doing and are confident that you made the correct connections rather than relying on the tools for feedback to catch problems.

5. 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. What was unclear in this lab writeup? How would you change it to run more smoothly next time?
3. For each of the following facets, did your design pass DRC? ERC? NCC?
   - dpbitslice
   - zipper
   - datapath

Extra Credit

As you are probably aware by now, Electric has plenty of bugs and idiosyncrasies. A major goal of this class is to improve the stability and ease-of-use of Electric. Please email your bug reports directly to Prof. Harris in the format described in Lab Manual 1.