Introducton to CMOS VLSI Design (E158)

Harris

Lab 3: Multiplier Layout

In this lab, we will layout a 32-bit serial multiply controller and wordslice. In the next lab, you will assemble the entire datapath and connect it up to pads for a chip.

I. Controller Layout

Look in the E158/labs/lab3 directory on Charlie. You will find a mult_lab3.jelib file containing a working version of Lab 2 and a partial layout of the multiplier.

The first step in this lab will be to finish the multcon layout. The layout has two parts: a 6-bit counter datapath and a random control block. The datapath has a regular structure, while the control is wired haphazardly. Part of the control is already completed, but the flip-flops and A2O1 gate are missing. Study the layout and compare it against the schematic until you understand how they relate.

The advantage of datapaths is that they can be densely wired over the cells where the structure exists to make the work worthwhile. The advantage of random logic is that it is easier to wire up when there is no systematic pattern to exploit.

Finish the layout. Instantiate six flopenr_c_2x cells to finish the 6-bit counter datapath. Place them next to the six adder cells so that the ends of the metal power and ground wires abut. Connect all of the inputs and outputs. Notice that the count[5:0] and nextcount[5:0] signals should be run on top of the cells in horizontal metal3 because there is no metal3 in the cells. The metal3 wires should be 6 λ wide, centered on multiples of 10 lambda (e.g. 0, 10, 20, 30, 40, …, 90). Use vertical metal2 wires to connect the metal3 to the exports on the cells. Note that wiring over the cells takes some practice because Electric tends to connect to things you didn't intend. It can be helpful to put contacts off to the side not on top of a cell, then wire to them, then drag them back to the position over the cell. It can also help to make cells hard to select so that you aren't accidentally selecting them. Don't forget to connect all the power and ground wires.

Then place the flopr_c and a2o1 cells at the end of the row of random logic. For ease of wiring, the random logic is connected together using horizontal metal1 and vertical metal2 wires that are above the cells rather than on top. Place the metal1 on an 8 λ pitch, similar to the metal already used.

When everything is connected together, the cell should look something like the one below. Be sure it passes DRC, ERC, and NCC.

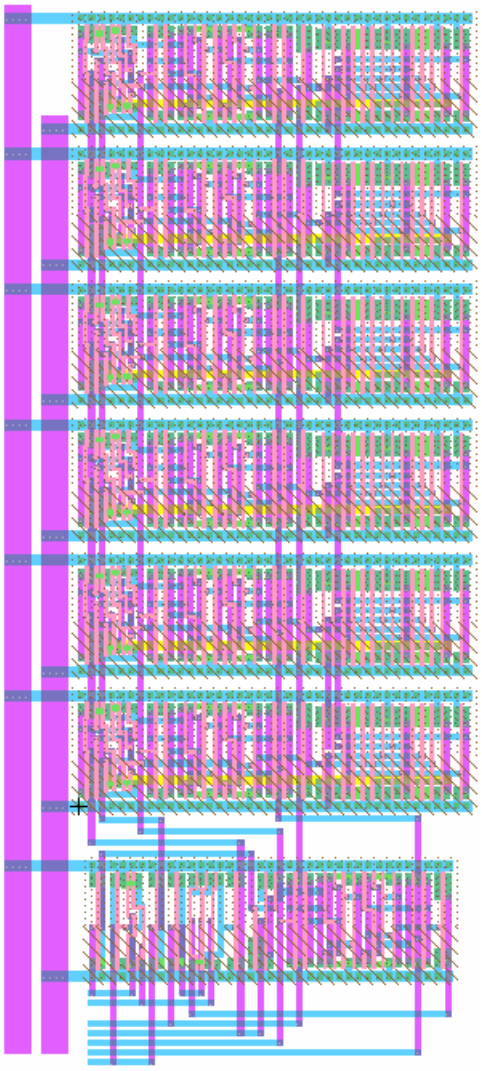

Figure 1: multcon{lay}

II. Wordslice Layout

Look at flopenr_1x_32{lay} to see a typical wordslice layout. Bit 0 is at the bottom and bit 31 is near the top. Above bit 31 are the zipper cells. The height of each bitslice is 102 λ. The inputs and outputs are exported. Flop bit *i* has a d[i] and q[i] export. On the zipper are exports for reset, en, ph1, and ph2.

It is possible to create a layout like this by manually instantiating 32 copies of a cell, wiring them together, then manually exporting all of the inputs and outputs as you did for the counter datapath. This is tedious and time consuming. Electric has some capabilities that help. These include arrays, autostitching, and mimic stitching.

Create a wordslice layout for the mux2_1x_32{lay}. Create the new layout cell. Place an instance of mux2_1x{lay} from MuddLib. (Always instantiate layouts and icons

rather than cutting and pasting. When you instantiate, you keep a link to the master cell. If the master has to be corrected, all the instances will automatically be corrected.) Position the cell at 0,0; you can do this by double-clicking on the cell to bring up node properties and setting the X and Y positions to 0. Export d0 as d0[0], d1 as d1[1], and y as y[0] with the correct input/output polarity.

Sometimes it is easier to view the cell as a box with ports, while other times you'll want to see what is inside the cell. Select the cell, then choose Cells * Expand Cell Instance * All the Way to view the contents. Unexpand Cell Instances does the reverse.

Next, array the cell 32 times. First, configure the array command so that it replicates and autoincrements the exports. Choose File * Preferences. Select the General category, then Nodes. Check the "Duplicate/Array/Paste copies exports" box, then click OK. Then choose Edit * Array. Set the Y repeat factor to 32 to make a column with 32 cells. Click the "Space by centerline distance" button and set the Y centerline distance to 110. Then choose OK. You should see 32 copies of the mux. The zeroth copy is the one you originally placed. The 1st copy should be at (0, 110) and should have inputs d0[1] and d1[1]. The second copy should be at (0, 220) with d0[2] and d1[2], and so forth. Our convention will be for the least significant bit to be at the bottom of the wordslice.

Next, connect the select (s) signals together with one long column of metal2. This could be done by manually drawing 31 arcs connecting the ports on each multiplexer. But Electric has an easier technique called Mimic Stitch. First configure Mimic Stitch by choosing File * Preferences. Select the Tools category, then Routing. Under restrictions, check "Ports must match" and uncheck the others, then click OK. Choose Tools * Routing * Enable Mimic-Stitching to turn on the feature. Now, zoom in until you can see the s port on copies 0 and 1 of the multiplexers. Under the Components tab, be sure that the metal2 arc box is selected so that Electric creates wires in metal2 rather than metal1 when both are possible. Left click on the s port in multiplexer 0. Right click on the s port in multiplexer 1 to create a metal2 arc between them (be sure that the arc appears in the proper column; it is easy to be misaligned by 1 λ from the via). Electric will automatically repeat this wiring 30 more times to multiplexer 1 to 2, 2 to 3, and so forth on the matching port (s). The messages window will say MIMIC ROUTING: Created 30 arcs. Now do the same with the sb port. Finally, choose Tools * Routing * Enable Mimic-Stitching to deselect mimic stitching.

The next step is to create the zipper. Recall that the multiplexer zipper contains an inverter and buffer to drive s and sb. Create an instance of invbuf_4x{lay} at (0, 3520). Reexport the s input of the invbuf as s. Connect the sb_out output to the sb column for the multiplexer using metal2. Double-click on the arc and set the arc name to sb (it is always good practice to name all nets so that you can find them during debugging). Connect s_out to the s column using metal2. Name this net sbuf. The multiplexer and buffer were designed so that the select wires line up conveniently. (Note that the column should not be called s because it would short by name to the s input. Some tools such as a Verilog simulator would assume that the s input and the s_out wire were connected if

they were both named s; this is a tricky bug to hunt down and fix, and it causes simulation to behave differently than the actual circuit.)

At this point, the only thing missing should be the power and ground exports. Choose Export * Reexport Power and Ground. The message window should report 66 ports exported (power and ground for the 32 muxes and one zipper cell).

Run DRC. There should be no errors, but fix any if you find some. Run ERC; there should be no well errors. Run NCC. You will get errors because power and ground are not connected between the bits in the wordslice. We will later connect wordslices to form a datapath and wire up power and ground for the whole datapath, which will fix the problem. In the meantime, choose Tools * NCC * Add NCC Annotation to Cell * Exports Connected by Parent VDD (and the same for Parent GND) to tell NCC to ignore the unconnected VDD and GND. Rerun NCC and you should get no errors. The summary should say that exports, topologies, and sizes all match. Fix any problems you might find.

What to Turn In

1) A printout of your multcon{lay}. Does it pass DRC? ERC? NCC?
2) Does your multiplexer wordslice pass DRC? ERC? NCC?
3) How long did you spend on this lab?