Harris

Introducton to CMOS VLSI Design (E158)

Lab 4: Datapath and Chip Assembly

In this lab, you will assemble the multiplier datapath and connect it up to the pads of a chip.

I. Datapath Assembly

Look in the E158/labs/lab4 directory on Charlie. You will find a mult_lab4.jelib file containing a working version of Lab 3 and a partial layout of the multiplier. You will also find the latest muddlib.jelib, the muddpads11_ami05.jelib pad library, and the multpads.arr pad frame arrangement file. Copy them to your directory and rename mult_lab4 to mult_lab4_xx.jelib.

The datapath can be viewed as a collection of 32 bitslices. The basic problem in datapath assembly is that wiring up all 32 bits manually would be painful and a waste of time. Instead, the preferred strategy is to carefully wire up one bitslice, then replicate that wiring 32 times. Electric provides some support to make this possible, but it is nevertheless tricky. In this lab, you will learn to do some simple datapath wiring. The mult{lay} cell is missing the multiplexer that you have designed. Your goal will be to add it to the layout and wire up the inputs and outputs.

Look at the mult{lay} and see how it is put together. You may find it helpful to go back and forth between expanding and unexpanding the cells so that you can make out the wiring over the cells and then see how this wiring connects into the cells. Metal3 wires are run horizontally, centered on multiples of 10 λ. Metal2 is run vertically, centered on the same grid as in the standard cells. 8-λ wide metal1 power and ground wires connect to the wide vertical metal2 power and ground straps on the left side. Most connections are made entirely within a bitslice, but some run between one bitslice and the adjacent one to perform shifts. Study the layout until you fully understand how it relates to the schematic; you will need this understanding to successfully do the next step.

The layout is missing the multiplexer wordslice that you created in the last lab. It should fit on the left side of the datapath, to the right of the power and ground straps. Instantiate the mux2_1x_32 and position it so that the power and ground wires abut with the adjacent prodlreg flip-flop wordslice. This can be difficult to see because the power and ground busses already run over the spot where the mux2 is going. However, the multiplexer is positioned correctly when there is a 4 λ space between the substrate and well contacts of the multiplexer and the adjacent cell. Double-click on the cell to edit the node properties and name it prodlmux to match the schematics. (Electric doesn’t care about the name, but it will make debugging easier if names are consistent.)

The next steps of adding wires are tricky, so choose Cell • New Version of Current Cell to create a copy. If you goof something up, you can revert to the old version and start wiring again.

The multiplexer has two 32-bit inputs, one 32-bit output, and one select input. Each bitslice needs two inputs and one output. According to the schematic, d0[i] should come from prodl[i+1] for all bits except the most significant. d1[i] should come from x[i]. And y[i] should connect to nextprodl[i]. We will make these connections for bit i = 0, then later copy them to all the other bits.

First, connect d1[0] to x[0]. x[0] should come in from the left on metal3, just like y[0]. It needs to connect to the metal2 multiplexer input, so create a m2m3 via. Place it in the same column as d1[0], centered on a row that is a multiple of 10 λ. Create an export on it named x[0]. Array the cell 32 times with a y centerline distance of 110 λ so that a contact is created on each bitslice. You should see the export increment such that the topmost one is x[31]. Then turn on mimic stitching. Make a connection between x[0] and d1[0]. You should see the a message in the Electric Messages window:

```
Wiring added: 1 Metal-2 arcs
MIMIC ROUTING: Created 31 arcs
```

This means that you drew one metal2 arc connecting the via to the port. Then the Mimic router created 31 more for the 31 remaining bitslices. When you are using mimic stitching, be careful to watch the messages window. Know how many wires you expect to have created. If you don’t get the right number, undo your stitching, figure out the problem, and try again. Otherwise, it is very easy to create a large number of incorrect connections that would be exceedingly painful to delete by hand and that probably will force you to start over with the routing. If you are getting the wrong connections, check the preferences box for routing and adjust how Electric decides which connections to mimic. Also note that while mimic stitching is on, Electric may mimic deletions. Be careful of this lest you delete far more wires than you intend.

You may wish to run DRC after making each connection to be sure that you haven’t caused a design rule error.

Next, connect y[0] of the multiplexer to d[0] of the adjacent flip-flop; this connection will be nextprodl[0]. A difficulty with this connection is that Electric tends to select cells when you really wanted to create wires over the cells. To get around this difficulty, you can make the multiplexer and flip-flop “hard to select” by invoking Edit • Selection • Make Selection Hard. Then you will need to click on the blue arrow on the icon bar below the Electric menus to explicitly select the cells; otherwise you will not be able to select them. Later, you can make the selection easy when you are done.

Make the connection using horizontal metal3. Place two m2m3 vias in the columns with the y[0] and d[0] signals you intend to connect. Connect the vias with a horizontal metal3 arc. Double-click on this arc and name it nextprodl[0]; this is not necessary for Electric, but helps in debugging. Select the two vias and the arc and array them 32 times. Connect the vias to the cell ports using vertical metal2. Then, while mimic stitching is enabled, connect the left via to y[0] and the right via to d[0]. Check that 31 additional arcs are mimicked on each connection.

Finally, connect d0[0] to prodl[1]. Create a m2m3 via in the same column as d0[0] on the row with prodl[1]. Array it 31 times instead of 32 because d0[31] does not connect to prodl[32]. Connect the via to d0[0] and make sure that the connection mimic stitches. Then connect the via to the prodl[1] pin and make sure it mimic stitches. You should see only 30 arcs created by each mimic stitch.

This completes the routing of busses. Next, route the remaining signals. d0[31] should connect to sumh[0]. Note that there is already a metal2 line bringing sumh[0] up to the top bitslice, so you should only need to make a horizontal connection in metal3. The s input at the top should connect to the start signal coming out of multcon. And VDD and GND should connect from the exports on the multiplexer to the metal1 busses running overhead.

When you are done, run DRC, ERC, and NCC on mult{lay}. Fix any problems. NCC is particularly tricky because a single mistake will lead to a large number of complaints. Your best defense is to thoroughly understand the connections you are trying to make and to get them right the first time. In a larger design, it is also helpful to use hierarchy so that there are only a modest number of word slices being connected at any given time. This reduces the number of places you could make a mistake.

II. 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 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.

Electric provides a handy pad frame generator that automatically assembles a pad frame for you from a library. To use the pad frame generator, you need your library, a pad library, and a pad arrangement file. The pad library is named muddpads13_ami05.jelib. The pad arrangement file, multpads.arr, is also in the class directory. Look at one of the pad arrangement files. It defines the library in which the pad frame is stored, the name of the cell to generate, and the name of your top-level design (called the core), and the ports that should be used to line the pads up together. It then contains a list of each pad for the system.

Open the muddpads13_ami05 library. Look at the schematic for pad_io (an input/output pad). Figure out how it works. Pad_in and pad_out are similar, but the enable is hardwired to 0 or 1.

Then look at the pad_io layout. You will see a big square pad consisting of a stack of metal1, metal2, and metal3, along with a region marked with circular stipples called the passivation layer. The chip is normally covered with a layer of silicon dioxide called the *overglass* or *passivation* to protect the circuitry. This overglass is etched away over the pad so that wires can be bonded to the metal stack. The “passivation” layer in Electric indicates where the etching should occur.

Before you generate the pad frame, be sure that mult_lab4_xx is the current library. Use File • Change Current Library if necessary to change it. Use the Tools • Generation • Pad Frame command and choose the multpads.arr file. This will create two cells: chip{lay} and chip{sch}. The cells will have a ring of pads around the periphery and the core cell in the middle. Generic unrouted arcs connect the pads to the core. Electric currently has a bug that prevents schematic busses from being connected to the pads properly; this will hopefully be fixed soon. Inspect the padframe layout and schematic to be sure each looks reasonable. Be sure to Expand Cell Instances all the way so that you can see the contents of the cells in the layout.

In a real design you would need to delete the generic unrouted arcs in the layout and replace them with real metal lines. You would also need to connect vdd and gnd to the pad ring using fat wires. This is called global routing and is sufficiently tedious that it is usually done with an automatic routing tool. We don’t have one, so in this lab you may skip that step rather than do it by hand. The overall layout is so big that Electric may display the multiplier core in grayed form on the screen. To see the contents, choose File • Preferences • Display • Display Control and adjust “Simplify objects smaller than” to 1 pixel.

III. 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.

If this were a real design, run the following pretapout checks to ensure the design is correct. Fix any errors. Because we haven’t completed the schematic padframe or hooked up vdd and gnd in the layout, the checks would generate some errors. Thus, don’t actually run these checks, but read through and understand why they are done.

1. File • Check Libraries
   Makes sure library is stable before verification proceeds.
2. File • Preferences • Tools • DRC • Clear valid DRC dates
   Do a fresh DRC on the entire chip, even if the cells passed once before
   (this shouldn’t be necessary, but is good for paranoia)

3. Tools • DRC • Check Hierarchically
   There should be no DRC errors
4. File • Preferences • Tools • NCC
   Make sure that Hierarchical Comparison and Check transistor sizes are selected.
5. Tools • NCC • Schematic and Layout Views
   If the pad frame schematic were complete, this should pass. But since The frame is incomplete, verify that NCC passes on the core.
6. File • Preferences • Tools • NCC
   Change to flat NCC and rerun NCC. This is just to be paranoid, lest Electric have a bug in hierarchical NCC. But again the pad frame is incomplete, so the test will fail.
7. Tools • Electrical Rules • Check Wells (this may take a while)
   If the power and ground pads were hooked up, this should give no errors.
8. Simulate the layout with Verilog using the testfixture used to validate the RTL.

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 and save the file as chip.cif. You will get a large number of resolution errors because Electric tends to create some arcs off of the ½ λ grid that CIF prefers. You can ignore these.

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.

This completes the lab. You now know how to create layouts and schematics. You know how to draw leaf cells, then organize them as datapaths or random control logic blocks. You know how to verify the design using DRC, ERC, NCC, 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!p

What to Turn In

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