Introduction to Extraction and the Custom Design Flow

EE241 Tutorial 3
Written by Brian Zimmer (2013)
Updated by Sean Huang (2019)
Updated by Daniel Grubb (2020)

Overview

In Lab 1 (GCD: VLSI's Hello World), you used the digital design flow to place-and-route a pre-existing library of standard cells based on an RTL description. In Lab 2, you pushed the same design through DRC and LVS. In this lab, we take an even closer look at a design. First, we will place-and-route a 4-16 decoder using the same Hammer flow from Labs 1 and 2. We will then extract the design and simulate both the non-extracted and extracted netlist. With that, we can compare the timing and energy results between pre- and post-extraction.

Then, we will take a closer look at the ASAP7 standard cells we have been using throughout the course and start a custom cell design in Cadence Virtuoso. You will layout a simple C2MOS latch, run DRC to verify the layout is manufacturable, and LVS to verify that your layout matches your schematic. Lastly, you will run parasitic extraction on your design and run some basic simulations to estimate some of its characteristics.

Getting Started: Decoder Extraction

We will once again start with updating our environment. On the instructional machines, you can either clone the same repository again or pull in the changes to an existing one.

```
  % cd /scratch/userA
  % git clone ~ee241/spring20-labs/ee241bS20
  % cd ee241bS2020
  % git submodule update --init --recursive
or
  % cd /scratch/userA/ee241bS2020
  % git pull            // you will need to reconcile any local edits you've made
  % git submodule update

  % source sourceme.sh  // NEEDS TO BE RUN EVERY NEW SESSION
```

Notice that there is now a decoder.yml in the lab repo. This contains the Hammer inputs for the 4-16 decoder we will be working with (it is pretty similar to gcd.yml in this case).

We will need to the the decoder through the tools using the Hammer flow. Again, since Hammer generates a helper Make include, we can just run the below command.

```
  % make lvs
```

If you take a look at the Makefile, there is something slightly different this time. Instead of calling **hammer-vlsi** directly while using the Hammer flow, we call **inst-vlsi.py**. This file contains a

HammerDriver that extends the base HammerDriver class. This is how the Hammer flow allows for flexibility in the design. If you want to edit the default steps that Hammer runs, you can do it like this. You may add a Hammer `hook` here to inject custom TCL that doesn't fit into a Hammer API (any real tapeout will certainly have some custom TCL). You may insert hooks before or after any default Hammer step (or another hook), you may replace a default Hammer step, or you may remove a Hammer step. In this case, we have removed a custom Hammer step that we don't need for this lab.

PEX: Parasitic Extraction

We will be using the Calibre tool PEX to run parasitic extraction. Parasitic extraction will produce a netlist with more accurately modeled capacitances and resistances for the nets in the design. We can then simulate this netlist with Spice to get more accurate results than just the original post P&R netlist. PEX will first run LVS (again) before exporting the extracted netlist.

We will start by opening up the decoder in CalibreDRV. Run the following from the ee241bS20 directory:

```
% ./build/lvs-rundir/generated-scripts/view_lvs
```

In CalibreDRV, click `Verification` then `run PEX...` on the toolbar. A dialog will open asking for a runset which you can exit out of. Click on `Rules`, then select **ee241bS20/pex-files/rcxControl_calibre_asap7_170801.rul**. This is the PEX rule deck for ASAP7. Then go to `Inputs`. For the Layout, PEX should extract it from the layout of the decoder it already has. For the Netlist, select **decoder.include.sp** in lvs-rundir (make sure "Export from schematic viewer" is unselected). Under `Outputs`, make sure the Netlist Format is set to `HSPICE`. Finally, click `Run PEX`.

Q: 1. Submit the first 40 lines of the "decoder.pex.netlist" file generated by PEX.

Q: 2. Simulate the decoder and measure the delay from A[3] rising (the rest are 0) to Z[8] rising and the average power for both the extracted netlist and the original LVS netlist to compare them (some tips below). State the output load capacitance you used.

We have supplied some HSPICE files to start with in **ee241bS20/pex-sims/** that have the testbenches mostly setup for you already. For the pre-extraction simulation, first run `make hspice-netlist` in the base directory. This will generate an HSPICE compatible netlist from the LVS netlist at `lvs-rundir/decoder.hspice.sp`. For this simulation, make sure that the ports match up with the instantiation of the decoder in decoder.sp (the ports can get messed up in the transformation). Make sure that the ASAP7 CDLs are included in your testbench (there are some provided helpers that already have this for you). The extracted netlist includes a few other files that will be produced in the lvs-rundir, so it might be easiest to run that simulation directly in the lvs-rundir.

Getting Started: Custom Design

In this portion of the lab, we will be exploring custom cell design. Figure 1 gives an overview of the steps involved in the design flow of creating a custom cell and integrating it into a VLSI flow. We will start by setting up Virtuoso (if you have not done so previously). We will import an existing

design of a flip flop in the ASAP7 PDK and introduce the steps you would take while building a custom standard cell. We will examine its schematic, export a netlist, run DRC and LVS, and simulate it. We will then discuss the remaining steps in the custom design flow to integrate your design into the VLSI flow we have been using. You may also work through a custom design of a C2MOS latch yourself for extra credit.

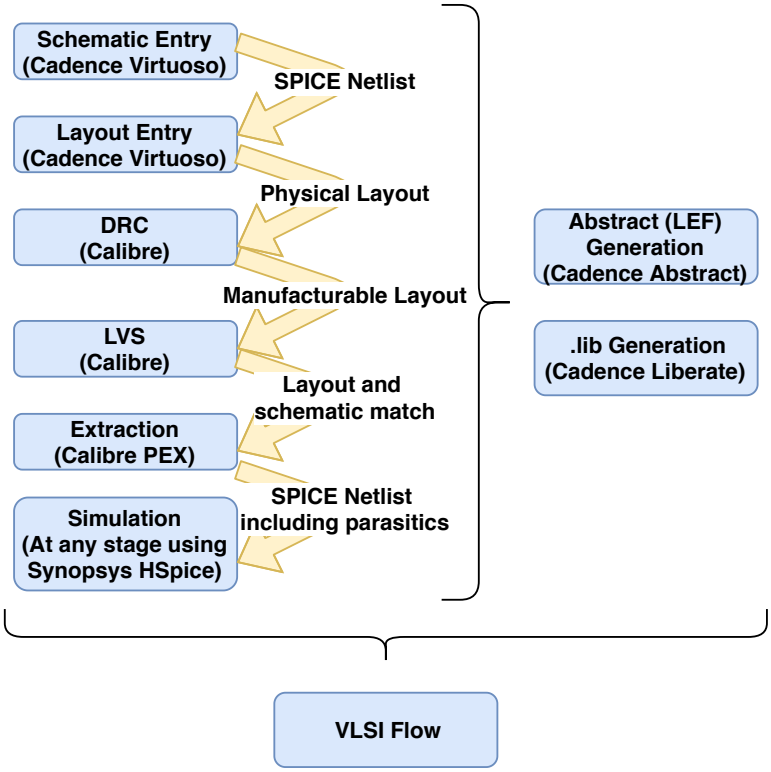

Figure 1: Custom Cell Design Flow

Start by running:

```
% source ~ee241/spring20-labs/asap7_virtuoso/sourceme.sh
% tcsh                \\ start C shell
$ source ~ee241/spring20-labs/asap7PDK_r1p5/cdslib/setup/setup_asap7.csh
$ exit                \\ exit C shell
% virtuoso &
```

If you look at the list of libraries, you will notice that there are several libraries already setup (in addition to the default virtuoso libraries). `asap7_TechLib` is the ASAP7 library that contains the different transistor flavors available in ASAP7. `asap7ssc7p5t` is another library provided by ASAP7 that contains a few example standard cell views, including a flip flop, inverter, and tapcell. If you wanted to view the layouts for the other standard cells in ASAP7, you could make a new library by going to File — New — Library. Call it (asap7_std_cells). Then select "Attach to an existing technology library" and choose asap7_TechLib. Then go to File — Import — Stream with Stream File as ee241/spring20-labs/asap7libs_24/gds/asap7sc7p5t_24.gds and select the asap7_std_cells library. Attach it to asap7_TechLib and click Ok. This will stream in the layouts for all the existing standard cells in ASAP7. If you go to the asap7_std_cells library, you will notice there are

only SLVT layouts. This is a quirk of ASAP7 in which the PDK doesn't provide different threshold flavor cells which is why the ASAP7 Hammer plugin will automatically hack the PDK to make the different threshold flavor cells available for use in a VLSI flow.

Creating the Custom Library

First, create a new library by going to File — New — Library and name it `custom_cell`. Then select "Attach to an existing technology library" and choose asap7_TechLib. We will be exploring the flip flop view that is already provided in ASAP7. In the library manager, go to `asap7ssc7p5t` and right-click on `DFFHQNx1_ASAP7_75t_R` and click `Copy`. In the new window, change the To library to be `custom_cell` and the Cell to be `custom_dff_R`, and click Ok. This will give us a pre-made schematic, symbol, and layout for our "custom" cell.

Schematic Entry

For a truly custom cell, you would create a new schematic view in your new library and build your design using the components in the ASAP7 PDK. Once you built it, you can simulate it to verify its operation, including Monte Carlo simulations to test its operation with process variations.

In this case, you can open up the existing schematic that we copied over (it should look like figure 2. Answer the following questions about the schematic:

Q: 3. What is the purpose of the tristate buffer that connect from the "MS" node to the "MH" node (output of the input latch)? Does this affect the setup time? Why do these tristate transistors have fewer fins than the input transistors?

Notice the top level pins of the schematic. These are the same pins that will be the inputs and outputs of our standard cell layout.

If you want to instantiate this cell in a testbench or other schematic, you need to create a symbol view. We already have one in this case, but to create one you can go to Create — Cellview — From Cellview, and click ok in the dialog box. You can leave the symbol as is, or use the drawing tools to draw something custom. When you are done, save and close the window.

Schematic Netlist Export

We will be running LVS/PEX against this schematic later, so we will need to export this schematic as a netlist. Most physical verification tools perform verification on generic filetypes, such as GDS files for layout and SPICE netlists for schematics. To export the DFF schematic, go to the Virtuoso main log window and go to File — Export — CDL...

Populate the following fields:

- Library Name: Your new Custom Cell library
- Top Cell Name: custom_dff_R (or your top cell name)
- View Name: schematic
- Switch View List: auCdl schematic
- Stop View List: auCdl

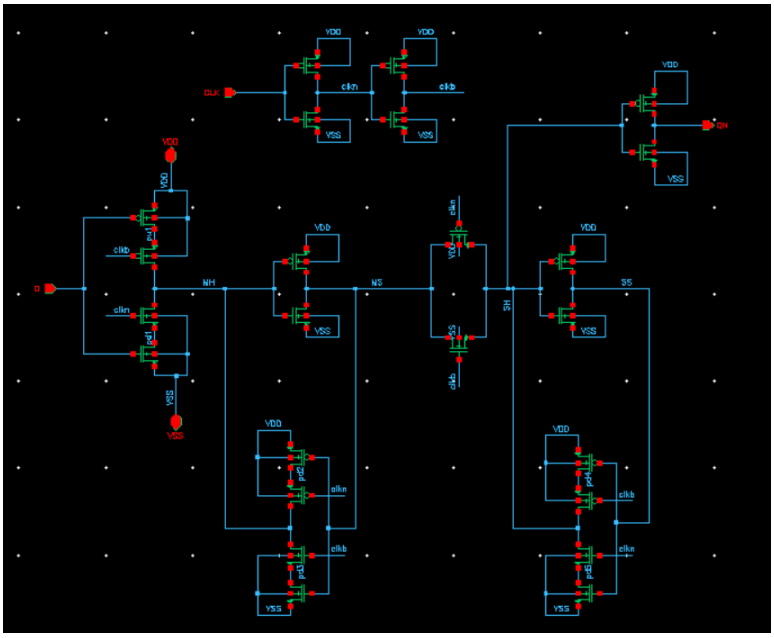

Figure 2: Schematic of the DFF.

- Output CDL Netlist File: custom_dff_R.cdl
- Make sure "Map Bus Name from <> to [ ]" is checked.

Hit Apply and wait for the CDL to be exported. Once this is done, you can look at the exported CDL to sanity check it.

Layout Entry

As discussed above, we will be using the existing DFF layout. The following instructions are a general tutorial for basic layout editing.

Figure 3 is a screenshot of what your copied DFF layout should look like. A great resource for layout in ASAP7 can be found here. This describes most of the important layers and provides a layout tutorial for a gate in ASAP7. If you are unfamiliar with transistor level layout, it would be instructive for you to go through the DFF layout and correlate parts of it to the schematic. If you're unsure of where to start, you can use the top level input pins (M1 pins) and the number of fins for different transistors as references.

For simple things, you only need a few commands to change the layout. By pressing ~, 1, 2, 3 etc you can show only certain layers. Press Shift-1 to add M1 to whatever is visible. To move something, hover over the object, then press "m" on the keyboard. To make something bigger or smaller, hover over the edge of the object, then press "s" on the keyboard, then click again at the final location. Press "Esc" if you selected something incorrectly. To add new wire, press Ctrl-Shift-W (then F3 will change the options such as the width). To just add a rectangle, select the desired layer in the layer pallete, and press "r". Then click on the two corners to create. Remember that selecting an object and hitting "q" will bring up info about it and allow you to edit certain

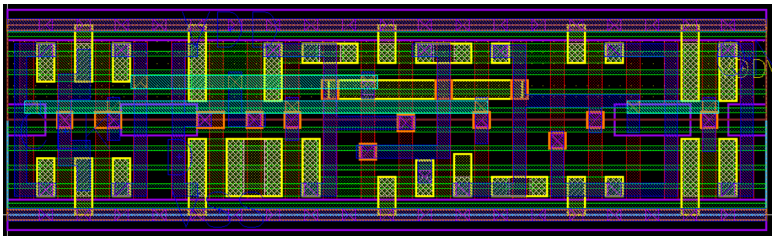

Figure 3: DFF Layout

characteristics. "k" allows you to place rulers to measure distances in the layout.

No modifications are needed for our layout, but you should now be familiar with what the DFF looks like.

Cell Verification

When you are done creating your schematic (and exported a netlist) and the layout, you must run DRC and LVS to verify that you have made the cell you wanted to make and that it is actually manufacturable. You can run Calibre directly from Virtuoso because there is a Calibre plugin (accessible from the tool bar) included in our setup. This lets us use the same Calibre tools we ran in the previous labs to verify our custom cell.

DRC

To run DRC, go to Calibre — Run nmDRC. Select the rule file to be ("/home/ff/ee241/spring20-labs/asap7PDK_r1p5/calibre/ruledirs/drc/drcRules_calibre_asap7.rul"). Make sure that under Inputs, "Export from layout viewer" is selected and then Run DRC.

Your design should be DRC clean except for a LUP error.

Q: 4. Read this DRC error and explain what the error is. What is the purpose and operation of a tapcell? You can look at an ASAP7 tapcell layout in the the same `asap7ssc7p5t` library we copied the DFF schmatic and layout from.

LVS

You don't have to run LVS for this lab, but if you are building a custom cell from scratch, you can do it similarly to DRC by going to Calibre — Run nmLVS. Change the runset file to "/home/ff/ee241/spring20-labs/asap7PDK_r1p5/calibre/ruledirs/lvs/lvsRules_calibre_asap7.rul". Make sure to select the CDL we exported earlier as the netlist.

If you are running into LVS errors, Calibre will describe the discrepancies and allow you to highlight the offending nets just like when we ran LVS for a larger design.

Extraction

To run parasitic extraction in Virtuoso, open the layout view and go to Calibre — Run PEX. The settings for this extraction are essentially the same as earlier in the lab and the same edited ruleset from the ee241bS20/pex-files/ should be used. Make sure that you select the CDL we exported earlier for the Input netlist (and that "Export from schematic viewer" is unchecked). Again, make sure that the ouptut format is set to `HSPICE`. A set of PEX netlists should be produced like for the decoder.

Note that it will say that the LVS run fails even thought the PEX netlist is produced. This is because it does not like that the substrate pins are also named VDD and VSS, so it can be ignored. In the generated pex netlist, the pins for the transistor body connections will be called VDD__2 and VSS__2.

Extracted Simulation

Finally, we will do a couple more simulations on the exported CDL and the extracted netlist. You should be able to use a very similar testbench to the one used earlier for the decoder. We have included a spice testbench in `ee241/spring20-labs/asap7_virtuoso` that may be useful to you. Remember that the output of this flip flop is QN, not Q. Make sure to check the ordering of the ports of your DFF between the different netlists.

Q: 5. Estimate the CLK-Q time of your DFF from for both the exported CDL and the extracted netlist from PEX. Include a screenshot (or the text) of your spice/otherwise testbench. State the load capacitance and rise/fall times you used.

Q: 6. Estimate the setup time of your DFF for both the exported CDL and the extracted netlist from PEX. State the load capacitance and rise/fall times you used. Also, include a screenshot of the waveforms of CLK, D, and QN for the simulation that you determined the setup time from (if you're using hspice, you can use waveview or wv on the command line).

If you're curious, you can try comparing your results to the timing parameters for DFFHQNx1_ASAP7_75t_R in `~ee241/spring20-labs/asap7libs_24/lib/asap7sc7p5t_24_SEQ_RVT_TT.lib`. You can find the table units and templates at the top of the file, and then find the timing lookup tables for clk-q, setup, etc. by searching for the DFFHQNx1_ASAP7_75t_R entry.

Extra Credit Layout

For some extra credit, you can run through the flow described in this document but create a basic dynamic (or static if you like) C2MOS latch in ASAP7 from scratch. Submit the schematic, layout, and DRC/LVS results. You can also simulate after extraction like we did for the copied DFF. You can do this from scratch, start from a gate an existing gate like a NAND2, or use the DFF layout and delete portions and reconfigure it.

Conclusion

This lab was meant to bring you a bit closer to the actual hardware you are designing with in a VLSI flow. The physical circuits you work with in a VLSI flow may sometimes seem quite abstracted away by the tools, but having the ability to look at the transistor level and understand the design is critical to solving problems that come up. The introduction to custom cell design is also meant to have you take a closer look at the standard cells we have used throughout the semester and understand the underlying circuits. One follow up activity to this would be to run a Monte Carlo analysis with your schematic to analyze the effect of process variations.

In order to fully use your custom cell in a digital flow, you would also need to generate all of the required collateral for the tools from your layout and schematic that we discussed in Lab 1. You would need to describe the cell's pre-computed timing characteristics (so it can be synthesized) and physical attributes (so it can be place-and-routed). If you want to explore this and try to add a custom cell to your ASAP7 flow, you can look at tools like the Cadence Virtuoso Abstract Generator (to create the LEF) and Cadence Liberate. Once you've done this, you can include it in your VLSI design flow for the synthesis and P&R tools to use!

Acknowledgements

Thank you to ECE 6332 - Introduction to VLSI Design (for the 21st Century) at UVA for the ASAP7 layout reference.