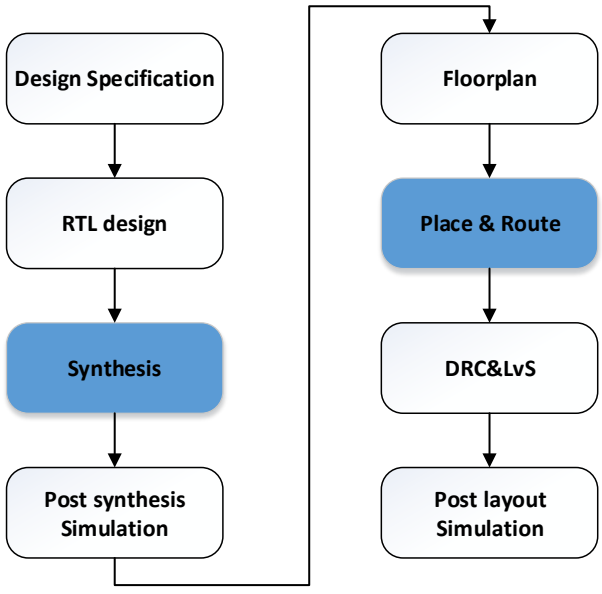
ECE 558/658 VLSI Design -- Fall 2021
Lab 4: Introduction to Synthesis and Place & Route

In all prior labs, you've designed your circuit at transistor level and created its layout; this style of custom design requires high engineering effort and is used sparingly. Large circuits with billions of transistors instead rely heavily on design automation to translate a design from hardware description language through synthesis and placement and routing into a layout made of standard cells from a library.

In this part of the lab, you will use design tools to go through an ASIC design flow to synthesize Verilog RTL and then perform physical design. These steps introduce powerful design automation flows that allow modern VLSI designs to integrate billions of transistors. Synthesis and P&R are usually automated with scripts instead of clicking buttons in GUI since these steps are iterated many times during a large design. The scripts you use in this part of the lab are modified from an NCSU tutorial at: http://www.eda.ncsu.edu/wiki/Tutorial:Place_%26_Route_Tutorials

The following figure shows a simplified ASIC design flow which starts from design specification and generates a GDSII file for fabrication. In this lab, we will focus on synthesis and place & route with standard tools that are widely used in industry. The RTL code be synthesized using the Synopsys synthesis tool -- Design Compiler. The generated netlist will be mapped to a physical implementation by the Cadence Encounter tool. We will be using Nangate 45nm Open Cell Library in this lab, which is based on the FreePDK45 physical design kit (PDK). The lab is split into 3 parts:

- In Part A you will learn how to run Synthesis (Design Compiler) and Place & Route (Encounter).
- In Part B you will compare the accumulator from part A against your custom layout accumulator from Lab 3.
- In Part C you will solve a few small problems to explore the behavior of the synthesis tool.



Follow the provided step-by-step instructions to get familiar with the tools. Be sure to include each required item (indicated by **SUBMIT**) in your report. You must also explain what you did and why; images alone are not sufficient.

Part A: Tutorial for Standard Cell Design (Synthesis and P&R) using 4-Bit Accumulator

Step A1: Synthesis using Synopsys Design Compiler [15pts]

1. On vlsicad, create a directory called lab4. Copy the source files from lab4.tar.gz into that directory, and then change directory to lab4.

```
>tar -xf lab4.tar.gz
>mkdir lab4
>cp -r lab4_files/* lab4/
>cd lab4
```

2. Enter Design Compiler shell by invoking the following command

```
>dc_shell
```

3. Read setup.tcl file by executing the following tcl file. However, you'll first need to modify this file slightly, so that it sets the "clkname" and "modname" variables for the design you are synthesizing.

```
>source setup.tcl
```

SUBMIT: What is the statement "`set modname`" used for and how did you change it?

SUBMIT: Why do you think the specification of clock name is necessary?

4. Read in Verilog files as design input by executing the following tcl file in dc shell. Pay attention to log information in standard output. Compilation errors would be reported here.

```
>source read.tcl
```

5. Constraints are very important inputs for synthesis tools. Browse the contents of the constraints file, and then execute the following tcl file to load the constraints into Design Compiler. If errors are reported, you must fix them.

```
>source Constraints.tcl
```

SUBMIT: If you want to achieve optimal area, how would you specify area constraint?

SUBMIT: The synthesis tool needs a target clock frequency to perform performance optimization and timing analysis. How do you specify the frequency in your constraints?

6. So far, we've setup all conditions for Design Compiler, and we can now invoke the synthesis command. Run the following tcl file. Pay attention to the log information in standard output. Fix errors here if necessary. You can ignore warnings for this tutorial example.

```
>source CompileAnalyze.tcl
```

7. Check the outputs. Synthesis generates an area report, timing reports and a netlist in Verilog format. You can find these files in current directory.

cell_report_final:	area report
timing_max_slow:	initial timing report without hold time fixing
timing_min_fast_holdcheck:	timing report for hold time check
timing_max_slow_holdfixed:	critical path report for slow corner
timing_max_typ_holdfixed:	critical path report for typical corner
timing_max_fast_holdfixed:	critical path report for fast corner
<module-name> _final.v:	the netlist

SUBMIT: Does your circuit have timing violations? (Tip: you may be able to find the answer in the report file "`timing_max_slow_holdfixed_tut1.rpt`")
SUBMIT: What is the estimated area of your circuit? How many flip-flops does your circuit use?
SUBMIT: Look at the netlist file which will be used for physical implementation. What primitives are used? Are the primitives part of the standard Verilog language, or from a technology-specific standard cell library?

8. The above steps can all be put in another tcl file and run by Design Compiler in batch mode as shown below. Use this approach to resynthesize the design with different target frequencies. Find the maximum frequency target that can be achieved (Tip: try changing parameter "`CLK_PER`" in `Constraints.tcl`. Once the frequency is unreasonably high for this design, it will cause a violation in the report file "`timing_max_slow_holdfixed_tut1.rpt`"). Once you've found the maximum achievable frequency, check the area and save the reports from that case. Now, find a second (lower) target frequency that causes synthesis to report a different area. Compare the synthesis results from these two frequencies in terms of their area and the specific cells that are used in each.

```
>exit
>dc_shell -f run_synth.tcl
```

SUBMIT: What is the maximum frequency based on synthesis?
SUBMIT: What is the lower frequency that produced a different implementation?
SUBMIT: Compare the area and the cells used in these two implementations.

Step A2: Place and route using Cadence Encounter [15pts]

Now that synthesis is completed, in the following steps you will use Cadence Encounter to create the layout and routing connections for the synthesized design. You should use here the version of your design that was synthesized for the highest frequency target. The provided lab4.tar.gz includes two pdf tutorials for Cadence Encounter to help you get started. The file design.conf in the pr directory has the configurations that you will load into Encounter. You will have to make a small change to that file based on your design, so be sure to look through that file to discover and make the necessary change.

9. Enter the pr directory by executing the following command

```
>cd pr/
```

10. Invoke Cadence Encounter GUI by executing

```
>encounter
```
(do **not** use "`encounter &`" here)

11. Load the design (`File -> Import Design -> load (All files *) -> design.conf -> ok`). Zoom your view to fit in the screen. You will see the default floorplan with multiple tracks.

12. Place standard cell `(Place -> Place Standard Cell… -> ok)`. When it is finished, you will see "Routed" on bottom right of the window. However, the placed layout doesn't show up on screen because it is still in floorplan view. To switch between views, you can click three buttons on the left of "online help" box.

SUBMIT: a screenshot in physical view without wire & via layers

13. Although the status shows it is routed, it is just a trial route to estimate how many metal layers are needed. There may also be routing errors reported (shown as red marks on layout). Perform detailed routing `(Route -> NanoRoute -> Route -> ok)`. This may resolve routing errors. If not, you need to fix them manually (Yes, tools are not perfect.) You should notice that there is some black space in the layout. The space has no gates placed, so it is not used. You can find information on the silicon area utilization in the logs. You can modify the provided `design.conf` file to create designs with a different aspect ratio.

SUBMIT: Implement the accumulator in a rectangular floorplan.
SUBMIT: A screenshot in physical view with wire & via layers.
SUBMIT: What is the silicon area utilization?

Part B: Comparing against Custom Design from Lab 3

Step B1: Comparison of Key Metrics [10pts]

Create a table comparing the design from Part A to your own design from Lab 3. You should specifically compare maximum frequency, power consumption, area, cell/row height, and the number of metal wiring layers used. Comment on the differences. You can use power and maximum frequency from Design Compiler synthesis reports and can find area, cell height and number of metal layers from Encounter. Note that the power and frequency numbers you obtain this way are estimates. You could obtain more accurate data by using extracted parasitics from the placed and routed design from Encounter, but we do not require this here.

SUBMIT: Table comparing results in terms of maximum frequency, power, area, metal layers used, and height of the cells in each row. Explain how you obtained each of the numbers.
SUBMIT: Discussion of what accounts for the differences, and the implications of those differences.

Step B2: Compare Critical Path [10pts]

Find the critical path that is reported by Design Compiler at highest target frequency. Compare it to the critical path you considered in Lab 3. Does the path from Design Compiler begin and end at flip-flops? How many stages of logic are on each path?

SUBMIT: Describe the critical path from Design Compiler and explain where you found it. Does it begin and end at the same place as your critical path from Lab 3? Explain.

Part C: Exploring Synthesis Results with Design Compiler

Step C1: Explore Area and Power Tradeoffs in Synthesis [25pts]

Modify the accumulator_ha Verilog module to create a 64-bit half-adder-based accumulator that you will synthesize for different frequency targets. You can reuse the Verilog and tcl scripts from earlier in this lab, but will need to make a few changes.

Synthesize the 64-bit accumulator for target clock periods of 1.5ns, 1.8ns, 2.0ns, 3.0ns, 10.0ns, and 20.0ns. After each synthesis, check whether the frequency target was met, and read the reports to find the cell area, dynamic power, and static power. There are many ways to find these results, including through the use of "report_power", "report_area" and "report_timing". Create the 3 plots described below. Note that power reported by Design Compiler will by default be at the target frequency; the 3rd plot (J/cycle) should include contributions from both static and dynamic power.

SUBMIT: Explain changes that you made to the tcl files and Verilog. How did you measure area/power?
SUBMIT: Create a scatter plot of cell area (y-axis) vs target clock frequency (x-axis)
SUBMIT: Create a line plot of total power (y-axis) vs target clock frequency (x-axis)
SUBMIT: Create a line plot of J/cycle (y-axis) vs target clock frequency (x-axis)
SUBMIT: For the 20.0ns clk period, run the "report_qor" command in dc_shell and include the results

Step C2: Explore Sizing and Performance Tradeoffs in Synthesis [25pts]

We saw in lab 2 how to upsize a path to drive a load capacitance. Now we want to see whether Design Compiler uses a similar sizing strategy to meet an aggressive timing constraint with a large load capacitance.

Create a Verilog netlist for the simple circuit drawn below. You will have to write the Verilog yourself but it is just a few lines; if you are not sure whether your Verilog is correct, you can synthesize it and check the post-synthesis netlist to see whether it matches your expectation. Remember that Design Compiler will need to know the module name, and will need to read your Verilog file, so you will have to make a few edits for that.

By default, Constraints.tcl specifies that the design being synthesized will have to drive a worst-case load of 4 D-flip flops (cell DFFR_X1). You will synthesize this design for much larger worst-case loads of 50, 100, 150, 200, 225, and 250 DFFs. Set the target clock period to 3ns in all cases. Run synthesis for each of these cases and check the synthesis results (e.g. report_area, report_timing), and view the final synthesized netlist to see which cells Design Compiler uses in each case. Report the results as described below. Note that the standard cell library includes DFFs that produce outputs in true and complemented form, and you may see either (or both) of those being used in some cases.

SUBMIT: Paste your Verilog module, and explain how you changed Constraints.tcl
SUBMIT: Include a table, where each row represents one of the output loads, that shows the area, whether timing is violated, and all cells used in the design. An example of the first table row is given below.
SUBMIT: Discuss whether you think Design Compiler is correctly applying the ideas of effort-based sizing
SUBMIT: Draw a diagram of the synthesized circuit from Design Compiler for the case of 250 DFF load

Load	Area	Timing violation?	Cells used
50	<number>	no	DFF_X1, NAND2_X1