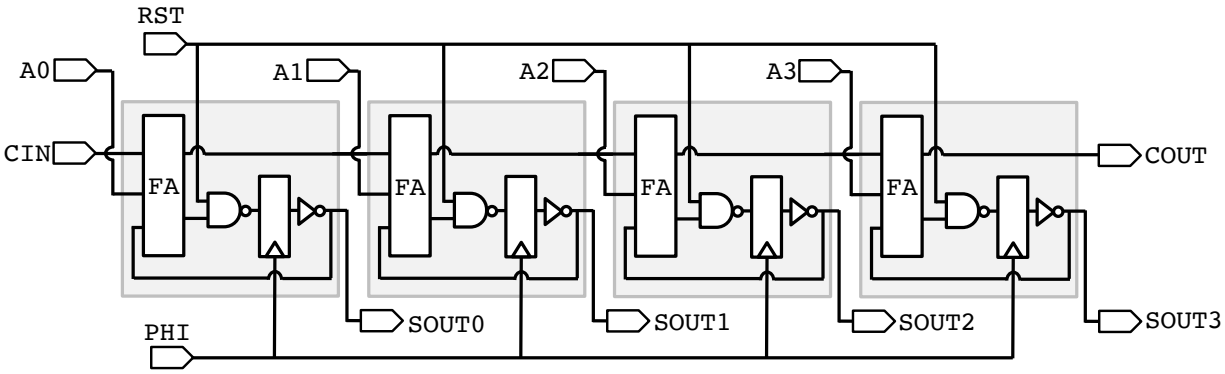
ECE 558/658 VLSI Design -- Fall 2020
Lab 3: Design and Analysis of a 4-bit Accumulator

In this lab you will create, in two different ways, a 4-bit accumulator circuit. Note that a fully functional lab 2 is a prerequisite for completing this lab. A 4-bit accumulator is a sequential circuit that in each clock cycle updates a sum by adding to it the current value of the input. Therefore, the sum stored in the accumulator represents the sum of the inputs since reset. Overflows are ignored in our accumulator.

As you have seen in the previous assignment, a one-bit accumulator consists of a full adder and a resettable register. A 4-bit accumulator is composed of four one bit accumulators. Therefore, the circuit for a 4-bit accumulator consists of a 4-bit adder and a resettable 4-bit register. Its 7 inputs are PHI, RST, {A3, A2, A1, A0}, and CIN. Its 5 outputs are {SOUT3, SOUT2, SOUT1, SOUT0} and COUT. The A and SOUT values each represent 4-bit words, in which A3 and SOUT3 are the most significant bits. The adder computes the sum of {A3, A2, A1, A0}, {SOUT3, SOUT2, SOUT1, SOUT0}, and CIN, and generates carry out COUT and a 4-bit sum that gets stored to flip-flops on the positive edge of the clock. Note that the braces { } in the above description are for readability of the 4-bit vectors and are not part of the signal names.

Students in ECE558 are using half adders instead of full adders. Their circuits are similar to the full adder accumulator, except that only CIN is being added to the sum in each cycle. In other words, the half-adder version of the accumulator computes the sum of all the valuations on CIN since reset.



In labs 1 and 2, you've designed your circuit at transistor level and created its layout. This style of design requires high engineering effort, and is used sparingly in large circuits. Large circuits with millions or 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. In this lab you will create the 4-bit accumulator using both styles of design. In Part A you will create the accumulator by instantiating the bitslice from lab 2. In Part B you will create the accumulator using automated synthesis and place-and-route 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.

There are two ways to complete this lab assignment: As in lab 2, there is an option to complete this lab without performing layout, for students that are having trouble accessing the tools remotely:

- Option 1 (layout) **performs steps A3 and A4,** and **skips A6 and B3.**
- Option 2 (no layout) **performs steps A6 and B3,** and **skips A3 and A4.**

All other steps are common to both options. If you are completing option 2, then the netlist you use in steps A5 and A6 will not be extracted from layout. Instead, it can be from step A1: either from the schematic editor or an HSPICE netlist that you've written manually (make sure to use subcircuits in HSPICE to keep the complexity manageable).

Part A: Custom Accumulator Design using Bitsliced Layout

Step A1: Schematic [15pts]

Draw a schematic for the entire 4-bit accumulator using the schematic editor tool (Cadence). Capture a screen image of the editor window. Use the following approach.

- Create a symbol view for the bitslice accumulator that you created in lab 2. Symbol views enable hierarchical schematics by hiding lower levels of detail (such as the interconnections of transistors). Symbol views are often simply rectangles with input and output pins, although many logic gates (inverter, NAND, etc.) and macro functions (mux, ALU, etc.) have customary shapes which you have drawn by hand many times.
- Instantiate the bitslice symbol four times to create the top-level 4-bit accumulator schematic. No individual transistors should be visible.
- The reset and phi inputs of all four bitslices should be connected.
- The carry out of each bitslice should connect to the carry in of the next bitslice to form a ripple carry chain.

SUBMIT: Image of your 4-bit accumulator schematic.

Step A2: Validate schematic using HSPICE .VEC [15pts]

Apply a test sequence that is based on the 8 digits of your student ID number. If you are in ECE658, you should convert each digit of the ID to a 4-bit value, and sum the 8 values by applying them on the {A3,A2,A1,A0} inputs of your circuit. If you are in ECE558, you should do the same, but using only the least significant bit of each digit and applying it on CIN. Don't worry if the sum overflows the counter. Make sure you reset the flip-flops before summing your digits.

SUBMIT: A table that shows your input sequence. Each row of the table should correspond to one cycle and show inputs, current state of flip flops, outputs, and next state of flip flops. The current state in the first cycle should be from reset.
SUBMIT: An image of the simulator showing inputs and outputs (waveforms). Annotate waveform to show that the inputs and outputs in each cycle match your table. Annotate to also show the reset occurring.

Step A3: Design a layout for your accumulator [5pts] (option 1 only)

Design a layout using Cadence Layout XL/L. Your layout must consist of four instances of the bitslice layout from Lab 2 connected by abutment; that is, no additional wiring may be used. Any changes you need to make should be made in the original bitslice layout so that you can instantiate it four times and get a clean and functional layout with no additional drawing. Do not "fix" the layout at the 4-bit hierarchical level. In the Cadence layout editor you can change the bitslice layout and have the change be automatically reflected in the 4-bit layout without having to re-instantiate.

- At bitslice boundaries, the Vdd and Gnd rails and the phi and reset signals should become connected automatically by abutment. Similarly, the carry out and carry in of adjacent bitslices should connect.
- Label the input and output pins at the 4-bit level. This is the only "new" content in your top-level layout.
- Since the aspect ratio of your bitslice layout was Y:X = 4:1, the aspect ratio of your 4-bit accumulator should be 1:1 (square).

SUBMIT: An image of your layout, with the total height and width annotated.

SUBMIT: Describe any changes you had to make to the bitslice layout from Lab 2.

Step A4: Testing [15pts] (option 1 only)

As you've done in the prior labs, run DRC to make sure your layout has no violations and LVS to make sure that it matches the schematic. Use Calibre PEX to extract a netlist that contains the parasitic capacitances and resistances from your layout. Reapply your test sequence from step 2 to be sure that your design works correctly.

SUBMIT: Screen capture of DRC, LVS results showing no violations.
SUBMIT: An image of the simulator showing inputs and outputs (waveforms).

Step A5: Determine Maximum Clock Frequency [15pts]

Identify the critical (slowest) timing path between flip-flops in your circuit which will limit the clock frequency of your accumulator (Hint: be sure to consider transitions that can propagate through multiple bitslices). Develop a test sequence that exercises the critical path. Use HSPICE to apply this sequence to the extracted netlist from step A4 and determine by trial and error the maximum clock frequency. Assume that the clock period can be set with 10 ps precision. As in lab 2, rise and fall times of all inputs, including the clock, must be kept at 10% of clock period as the clock period is changed.

SUBMIT: A description of the critical path, and discussion of why you believe it is the critical path.
SUBMIT: The test sequence which exercises the critical path.
SUBMIT: Image of the simulator output for your sequence, showing correct operation at maximum clock frequency. Annotate frequency on plot.
SUBMIT: Image of the simulator output when the clock period is 20% shorter than above. Annotate frequency on plot and highlight where the incorrect value is observed in the waveform.

Step A6: Measure Switching Power [10pts] (option 2 only)

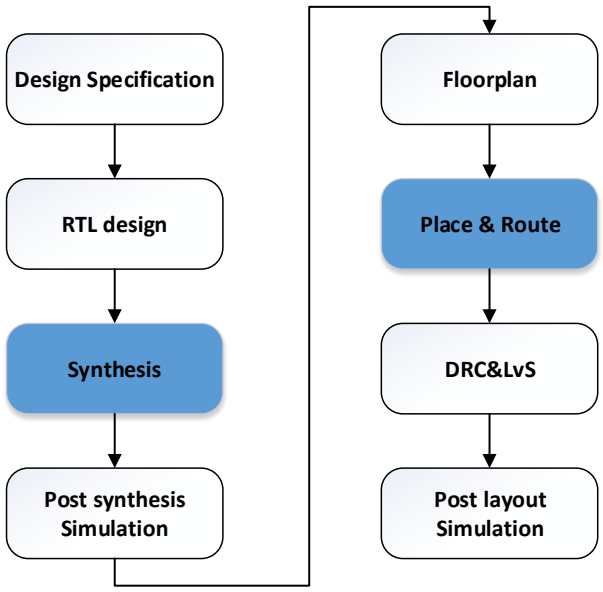
Find the average switching (dynamic) power consumption of the extracted 4-bit accumulator netlist when running your test vectors from step A2 at the maximum clock frequency from step A5. Because the power in this step is for a specific frequency, you can report power in units of µW instead of µW/MHz. Total power is the sum of switching power and static power, but note that in this step you are trying to specifically find average switching power. You can make the simplifying assumption that total power is equal to switching power if you can show that static power is less than 10% of total power. Otherwise, you must take steps to ensure that static power is not being counted toward average switching power.

SUBMIT: Average switching power and description of how you obtained the measurement.
SUBMIT: Describe how you handled static power. If you choose to neglect the contribution of static power, justify why this will not introduce more than 10% error.

Part B: Standard Cell Accumulator Design (Synthesis and P&R)

In this part of the lab, you will use automated design tools to go through an ASIC design flow to synthesize Verilog RTL code and then perform physical design. These steps introduce powerful design automation flows that allow modern VLSI designs of processors and SoCs 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 to adjust various aspects of the design such as clock frequency, area, and so forth. 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 will 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). Follow the provided step-by-step instructions to get yourself familiar with the tools and be sure to include each required item.



Step B1: Synthesis using Synopsys Design Compiler [15pts]

1. On vlsicad, create a directory called lab3_partB. Copy the source files from lab3.tar.gz into that directory, and then change directory to lab3_partB.

```
>tar -xf lab3.tar.gz
>mkdir lab3_partB
>cp -r lab3_files/* lab3_partB/
>cd lab3_partB
```

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 will be 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. If there were errors during compilation, they would be reported here.

```
>source read.tcl
```

5. Constraints are very important inputs for synthesis tools. Execute the following tcl file to load constraints into Design Compiler.

```
>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. The synthesis generates an area report, timing reports and a netlist in Verilog format. You can find these files in current folder.

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
accumulator_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: Take a look at the netlist file which will be used for physical implementation. Explain what is RTL synthesis based on your understanding so far. Is it technology dependent?

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 B2: Place and route using Cadence Encounter [10pts]

Now that synthesis is completed, in the following steps you will use Cadence Encounter for layout implementation of the synthesized design. You should use here the version of your design that was created for the highest frequency target. The provided lab3.tar.gz includes two pdf tutorials for Cadence Encounter to help you get started.

9. Enter the pr directory by executing the following command

```
>cd pr/
```

10. Invoke Cadence Encounter GUI by executing

```
>encounter
```

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 needed. And it is very likely that it reports routing errors (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 log information. 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?

Step B3: Synthesis for Different Bit-widths [10pts] (option 2 only)

Repeat the Design Compiler synthesis from step B1 using accumulators of different bit-widths. Change the Verilog file to create 8-bit, 16-bit, 24-bit, and 32-bit accumulators. For each bit-width, find the maximum achievable frequency and the area of the synthesized design at that frequency. Find also the area when the frequency is half of the maximum achievable frequency for each bit-width. Give the numeric value of the results in a table, and plot of frequency and area vs bit-width. Do these results make sense to you? Explain.

SUBMIT: Table of results.
SUBMIT: Plot of area and frequency vs bit-width.
SUBMIT: Discussion and analysis.

Part C: Comparing Results from Custom (from Part A) and Standard Cell (from Part B)

Step C1: Compare results [5pts]

Create a table comparing Part A to Part B in terms of maximum frequency, power consumption, area, cell/row height, and the number of metal wiring layers used. 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. Discuss the differences.

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

Step C2: Compare Critical Path [5pts]

Compare the critical path that is reported by design compiler at highest target frequency to the path you considered to be critical from Part A? Does the path from Design Compiler begin and end at flip-flops? The synthesized design may be different from your design, so you should not expect to necessarily see exactly the same gates on the 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? Explain.