Lab 4: 4-bit Carry Ripple Adders

In this lab, we will design and perform the layout of a *PG Carry Ripple Adder.* Our goal is to obtain a minimum-sized, reusable and scalable 4-bit carry ripple adder with a reasonable worst-case propagation delay. ***We will have two parts in this lab. The first one is the main requirement, and the second one is a 50pts bonus.***

4-bit Ripple Carry Adder (Main Lab)

First, we are going to build a 4-bit Ripple Carry Adder shown in figure 11.12 in the textbook. The same figure is provided below:

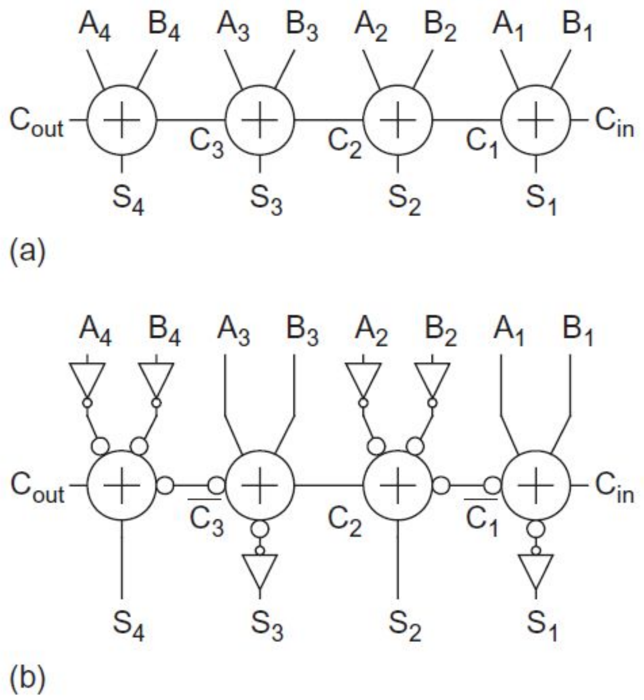


FIGURE 11.11 4-bit carry-ripple adder

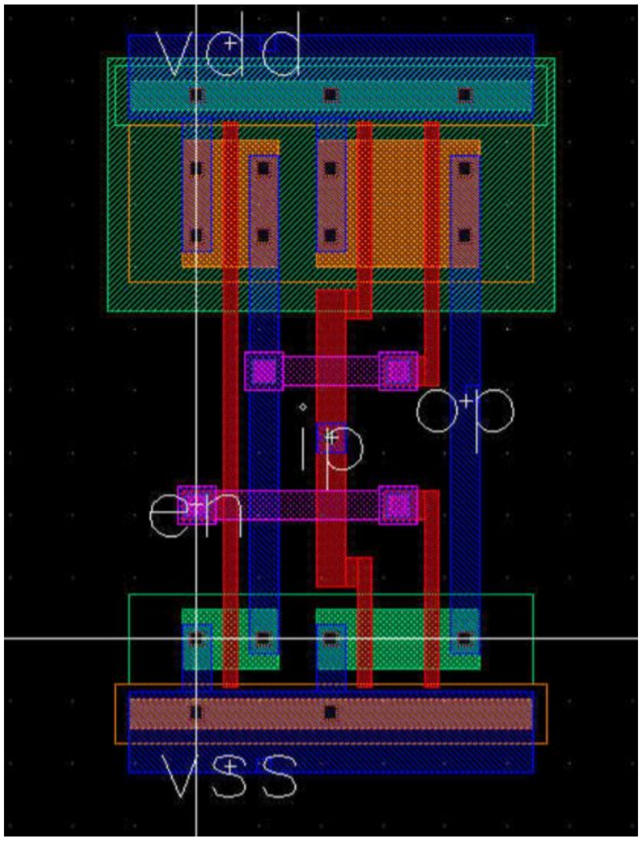
You are free to choose to implement the 1-bit full adder using any design mentioned in-class and in the textbook. You can use separate circuits for S and C_{out}, mirror static CMOS full adder, or transmission gate implementation. If you prefer CPL full adder or dual-rail domino logic, please make sure you understand how the precharge and evaluate phases work and note that the sizes may be harder to optimize.

Hierarchical VLSI Design

To implement the logic in such complexity, you are highly recommended to use *hierarchical design*. That is, build cell views for each small component and reuse them in both schematic and layout of a higher hierarchy module. For example, you can draw the schematic, symbol, and layout of the XOR, NAND2, AOI and INV individually and instantiate them in a full_adder cell. Then duplicate and put them in the 4-bit adder module. Use upper layer metals as wires to connect each cell and don't forget the well taps.

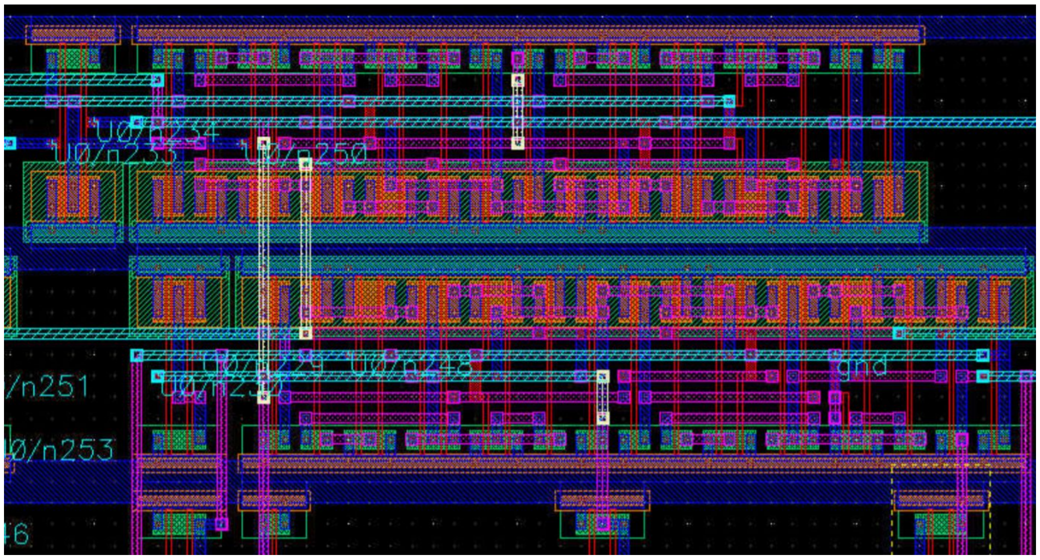
After learning through the previous labs, you must be comfortable with hierarchical design in schematics, since we've already used symbols for DUT in the testbenches. However, you may not know we can apply a similar technique to layouts. Open the layout editor and find Create->Instance. Then you can instantiate your previous designed layout and add it to the workspace. Hotkey <Ctrl>+F & <Shift>+F help us to toggle between the abstract view and the "x-ray" view. With an x-ray view, we can see through the layout of each instantiated unit but CANNOT EDIT it. That means when drawing the units, we better think of how they are going to be put in a larger design.

Some good rules to follow come from standard cell libraries, which requires a lot of reuse of basic cells. An example of the standard cell layout can be seen on page 23 in this PDF as well as the following:



As you can see, it has well taps right underneath the VDD/VSS rail, making it easier to bias the bodies. The designer also extend the n-well so when we want to put it side-by-side, the wells touch each other and become a big chuck automatically that we don't have to worry about how to connect them. It is a good trick to apply to power/ground lines as well.

From the following figure came from the same PDF on page 25, it shows the merit of following this sketching style. Not only can we put one cell on the side of another, we can also flip it upward and create another row. VDDs then can be easily connected by putting a big rail of M1 on top of them. The same rule applies when we need more row, when GNDs could overlap. We can also see other signals connected through upper layer metals (M3/M4).



Two adjacent rows of cells

Main Lab

Part I: Pre-Layout Design (50pts)

1. [30pt] Schematic of a 4-bit Carry Ripple Adder Cell
   Please draw the schematics of a 4-bit carry ripple adder. You are free to choose the structure of the inner cells as long as the final logic is correct. Attach a screenshot of all components you design, including the final 4-bit adder and the submodules in the circuit. You need 14 pins for the adder:
   - A1 ~ A4
   - B1 ~ B4
   - S1 ~ S4
   - C0, C4

2. [20pt] Test Bench and Pre-Layout Simulation
   Create a testbench for the adder and give at least one example input to justify the correctness. We do not ask for a specific input and loading setup this time. However, please note that you are going to reuse this circuit to build a 4x4 multiplier in the final project. It is recommended to have a proper testing environment to check if it works with other building blocks. For example, a unit/double-sized buffer (two inverters back-to-back) as the input and regular fan-out-four unit inverters as load is a good point to start.

Part II: Layout, Post Simulation & DFF Characteristics (50pts)

1. [40pt] DFF Layout
   Draw a layout for the adder. **You can only use three metal layers (M1, M2, & M3) in this lab.** Attach a screenshot of your design including rulers showing the size of the *bounding box*. If you are using the hierarchical design method, attach screenshots of all units. No DRC, LVS screenshots needed for every unit. Only include them for the final adder cell.

   Again we do not have an area-delay competition this time, but you might want to use the adder in your final project. Try to minimize the area and plan the IOs beforehand.

2. [10pt] Post-Layout Simulation & Worst-Case Propagation Delay
   Repeat the simulation in part I with the extracted model. This time, try to come up with an input sequence that can triggers the worst-case delay. State how you did it and include the numbers in the report.

Bonus PART (50pts)

4-bit PG Ripple Carry Adder

As a bonus, you can build a 4-bit PG Ripple Carry Adder shown in figure 11.12 in the textbook. The same figure is provided below:

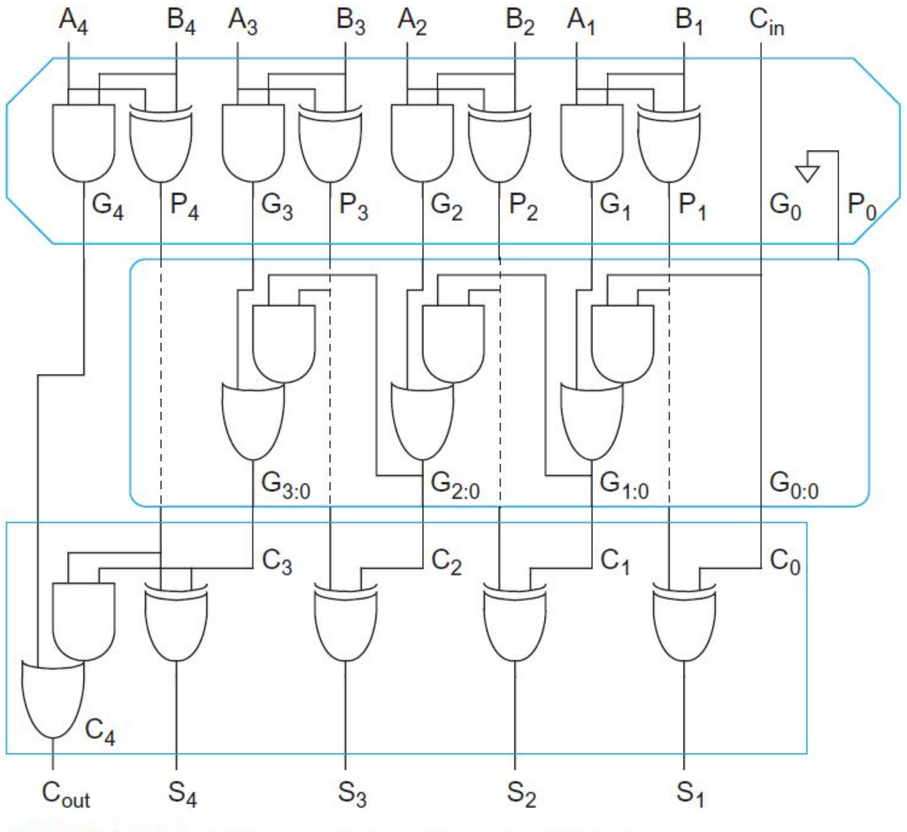


FIGURE 11.14 4-bit carry-ripple adder using PG logic

From the top to the bottom, there are three logic groups, bitwise PG logic, group PG logic, and sum logic, respectively.

- Bitwise PG logic produce the *propagation* and *generate* for each bit using following equations:

$$G_{i:i} \equiv G_i = A_i \cdot B_i \qquad G_{0:0} = C_{\text{in}}$$
$$P_{i:i} \equiv P_i = A_i \oplus B_i \qquad P_{0:0} = 0$$

- Group PG logic in the middle combine PG signals to determine group generates using:

$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j}$$
$$P_{i:j} = P_{i:k} \cdot P_{k-1:j}$$

- Sum logic calculate the sums using:

$$S_i = P_i \oplus G_{i-1:0}$$

In figure 11.14, it uses noninverting gates. That often leads to more stages of logic. The following Figure 11.16(b) shows how to alternate two types of inverting stages on alternate rows of the group PG network to remove unnecessary inverters. You are free to use the original logic in figure 11.14 or follow the rule shown in Figure 11.16(b) to modify the PG network. (Refer to the textbook for notations.)

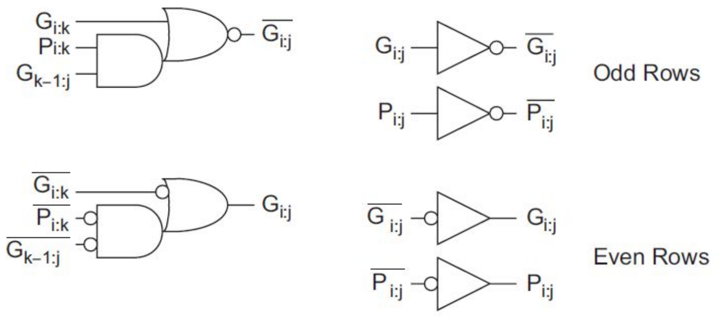


Figure 11.16 Group PG cells

Bonus Requirements

Part I: Pre-Layout Design (25pts)

1. [15pt] Schematic of a 4-bit PG Carry Ripple Adder Cell
   Please draw the schematics of a 4-bit PG carry ripple adder. The recommended structure is shown above. You are free to adjust the inner cells as long as the final logic is correct. Attach a screenshot of all components you design, including the final 4-bit adder and submodules in the circuit. You need 14 pins for the adder:
   - A1 ~ A4
   - B1 ~ B4
   - S1 ~ S4
   - C0, C4

2. [10pt] Test Bench and Pre-Layout Simulation
   Create a testbench for the adder and give at least one example input to justify the correctness. We do not ask for a specific input and loading setup this time. However, please note that you are going to reuse this circuit to build a 4x4 multiplier in the final project. It is recommended to have a proper testing environment to check if it works with other building blocks. For example, a unit/double-sized buffer (two inverters back-to-back) as the input and regular fan-out-four unit inverters as load is a good point to start.

Part II: Layout, Post Simulation & DFF Characteristics (50pts)

1. [20pt] DFF Layout
   Draw a layout for the adder. **You can only use three metal layers (M1, M2, & M3) in this lab.** Attach a screenshot of your design including rulers showing the size of the *bounding box*. If you are using the hierarchical design method, attach screenshots of all units. No DRC, LVS screenshots needed for every unit. Only include them for the final adder cell.

   Again we do not have an area-delay competition this time, but you might want to use the adder in your final project. Try to minimize the area and plan the IOs beforehand.

2. [5pt] Post-Layout Simulation & Worst-Case Propagation Delay
   Repeat the simulation in part I with the extracted model. This time, try to come up with an input sequence that can triggers the worst-case delay. State how you did it and include the numbers in the report.

Submission

Please submit a ***tgz file*** of your library along with ***the written report*** to Canvas by **12/03 (Tue.) 11:59 PM**. *The schedule is tight, so make your report concise. The more important purpose is to make sure it works in your final project.* Good luck.

Tips for creating tarball:

```
 1) Go to the directory where you can see your library folder.
 2) tar -czvf inverter_PID.tgz name-of-your-library
```

Please replace PID with your PID and the name of your virtuoso library as name-of-your-library. For example, if your PID is andrew123, and your virtuoso library name for the assignment is proj1, then the command should be:

```
      tar -czvf inverter_andrew123.tgz proj1
```

And you should submit the file named inverter_andrew123.tgz.

Hint: if you're using [ssh.ece.vt.edu](ssh.ece.vt.edu) to access the software, you can use `<CTRL><SHIFT><ALT>` key combination to bring out Guacamole Menu, where you can upload and download files from the remote server by drag and drop.