ECE425: Introduction to VLSI System Design
Machine Problem 1
Due: Friday 11:59pm, February 25th, 2022

Introduction

This machine problem will introduce cell-based circuit design. You will create a miniature cell library implementing several basic logic gates. Then, you will implement a full adder by plugging the cells together. This will give you a feel for the impact of individual cell design and multiple cell integration on the quality of the overall layout. The cells will form a library, which may be reused for MP2. This handout gives suggestions on how to design and arrange the cells, but you may be more creative if you like. Please note any original ideas you have tried in your hand in!

In this MP1 you should follow the design flow listed below:

schematic design -> NC-verilog -> layout design -> DRC -> extract layout -> LVS

MP1 Report (10 points):

Single PDF file (mp1_*netid*.pdf) containing following items in order

1. **Basic cells**
    a. **xor2:** schematic and layout
    b. **aoi22:** schematic and layout
2. **Full adder**
    a. schematic
    b. simulation waveform
    c. top-level of the layout hierarchy
3. **8-bit adder**
    a. schematic
    b. top-level layout
    c. LVS output report (si.out)
    d. Simulation comparison output report (compare.out).
    (Save the comparison report as a text file, then combine to PDF file.)
4. Please also submit all your design files in a zipped folder (such that anyone can evaluate your work in cadence).

Part 1: Base cell design and verification

Schematic Design

The tutorials had you extend existing libraries, but now you should start with a blank slate. We will build our circuits on an academically available (through MOSIS at the University of Southern California) manufacturing process from TSMC (Taiwan Semiconductor Manufacturing Corporation) using a design kit from NCSU (North Carolina State University). It will allow us to draw transistors with a 240 nm nominal gate length and 360 nm minimum width, corresponding to the state of the art around 1997.
To use this kit in Cadence, open your *Library Path Editor* and make sure you have the following entries:

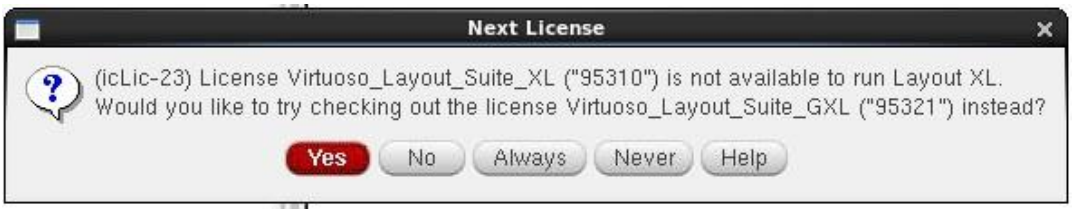
cd ece425.work *(please use the exact folder name: ece425.work)*
module load ece425-sp21
export CDK_DIR=/software/ncsu-cdk-1.6.0.beta
virtuoso &

Library	Path
basic	*${CDK_DIR}/lib/basic*
NCSU_Analog_Parts	*${CDK_DIR}/lib/NCSU_Analog_Parts*
NCSU_Digital_Parts	*${CDK_DIR}/lib/NCSU_Digital_Parts*
NCSU_TechLib_tsmc03d	*${CDK_DIR}/lib/NCSU_TechLib_tsmc03d*

(Be sure to start Cadence from your work directory (e.g., ~/ece425.work) and not any subdirectory. ALWAYS remember to FIRST run the command "module load ece425-sp21" before launching the Cadence software!)

1. To create your own empty library, select *File => New => Library…* in the CIW or Library Manager.
2. For the name, type *ece425mp1*.
3. Click *Attach to an existing tech library* and select *NCSU_TechLib_tsmc03d*. This is what associates your project with the TSMC process.
4. Press OK

If you encounter the following pop up window while opening any schematic, select “Always”.



5. Now you may select *File => New => Cell View* and use your knowledge from MP0 to create all the cell schematics. For the *Library Name*, select *ece425mp1*, which was created in previous step.
6. Please use the *nmos*/*pmos* transistors from the “**NCSU_Analog_Parts”** library and *vdd/gnd* symbols from the “**basic**” library.
7. Name the inputs as *a, b, c, d* (using only as many as necessary, of course) and the output as *y*. Now, implement these cells:

inv **nand2** **nor2** **aoi22** **xor2**

Please follow the names in this list – “2” indicates two inputs, and “inv” is short for “inverter.” See Weste & Harris pp. 12 and pp. 329 for the schematic of an AOI22 gate (and-or-invert with 2x2-input ANDs). Let *a* and *b* connect to a parallel chain of transistors in this AOI.

Create your “xor2” using exactly 10 transistors. (Hint: the solution includes an aoi21. To include an aoi21 in your xor2, you may first use Boolean Algebra to convert xor2 to a form including aoi21. You may use Google or Weste & Harris textbook for more clarification.)

Transistor sizing

The word *transistor* is short for *transferring resistor*, as the voltage at the gate terminal is “transferred” to the resistance between the source and drain terminals. The resistance of a resistor is proportional to its length and inversely proportional to cross-sectional area:

R = rho * length / (width * height) (rho: the resistivity of the material)

FETs are similar, except that height is adjusted by the gate voltage. (Height and length also depend on the source-drain voltage, but we can ignore that for now.) Length and width are controlled in the layout by the designer. We will keep all lengths to the minimum of 0.24 microns dictated by the process – which leaves only the width up to you.

Transistors may be connected in series, like resistors, with the resistances adding up. However, because the above definition of resistance is *very* approximate, we will avoid units of resistance. It is simplest to think of several transistors in series as equivalent to a single transistor with small width. Because

resistance varies inversely with width, two series transistors A and B will have:

$R_{total} = R_A + R_B$	(using resistance)
$1/W_{total} = 1/W_A + 1/W_B$	(using width)
$W_{total} = 1/(1/W_A + 1/W_B)$	(or with the *parallel operator*: $W_{total} = W_A \parallel W_B$)

The non-physical value $1/W_{total}$ is called the *effective width*, and when expressed relative to the minimum transistor width, the *drive strength*. Thus a 1/2x drive strength NAND gate in our process would have two 0.36 micron NMOS in series with an effective width of 0.36 / 2 = 0.18 microns.

A PMOS transistor has about twice the resistance of an NMOS of the same size. This is analogous to a higher resistivity material being used for a resistor. Therefore, the drive strength of any PMOS must be divided by two. A 1/2x drive strength NAND gate would have two 0.36 um PMOS in parallel, each with effective width 0.18 um or drive strength 1/2x.

Effective width is generally not affected by parallel connections, because transistors generally turn on one at a time. After the first transistor has turned on, it will finish driving the new output value before there is time for the second parallel transistor to switch. If their gates are connected together, however, they will switch at the same instant. In this case, they combine to form a single "super-transistor," with the widths adding up. Little distinction is made between a single large transistor and an array of smaller transistors in parallel – a schematic diagram will simply specify a total effective width without showing any parallel connections.

Please specify sizes for all the transistors in all your cells. Each cell should have drive strength equal to one minimum-size NMOS transistor, or effective width of 360 nm. (This also applies to the internal paths in the *xor2* and *and2*.) Fill in each transistor's width in its properties window. Note that logic gates based on pass transistors or transmission gates have no definite drive strength, so they will not be accepted.

NAND gate example sizing is illustrated in the figure below.

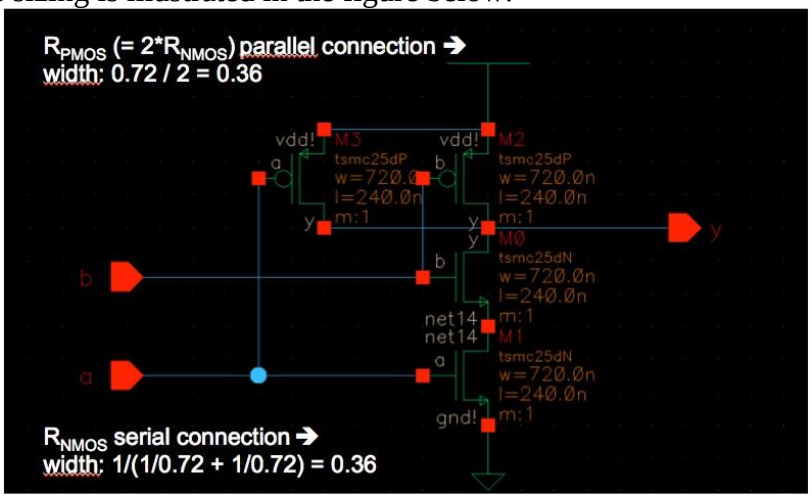


Figure. NAND gate pmos/nmos transistor sizing

Verification

Every step of circuit design is typically followed by an automated correctness check. Before starting layout, we will ensure that the schematics are correct using Cadence's NC-Verilog simulator. The simulator will apply a series of inputs to a virtual version of your circuit and allow you to see its internals in action. Verilog is used for logic verification (hence the name), so it will produce only 1's and 0's and consider the overall topology of your circuit's transistors. More precise physical simulation is used at later stages of the design process. As we are putting the smallest pieces of logic through the most approximate simulator, you will get your results instantaneously.

Note:

If your NC-verification failed due to license error which may be highly possible to occur, you may follow the instructions below to solve the problem:

1. save your schematic and close cadence first.

2. make sure your working folder is named **exactly** *as "ece425.work" and is located* **exactly** *at your home directory.*

3. remove the .cdsenv and .cdsinit file in the working folder

4. remove all the simulation folders (inv_run1, xor_run1 etc) you created before.

5. type ' cd ece425.work' to **go to ~/ece425.work/** *,then run the following command:*

module load ece425 *(module load ece425-sp21 does* ***NOT*** *work)*

6. check file cds.lib in your working directory, there are a lot of macros defines in the file. If the macro "ece425mp1" is not defined, add the following line at the end of the cds.lib file.

DEFINE ece425mp1 ~/ece425.work/ece425mp1

7. open virtuoso **exactly** *in your working folder. (run the command: virtuoso &)*

8. Retry the NC-verification

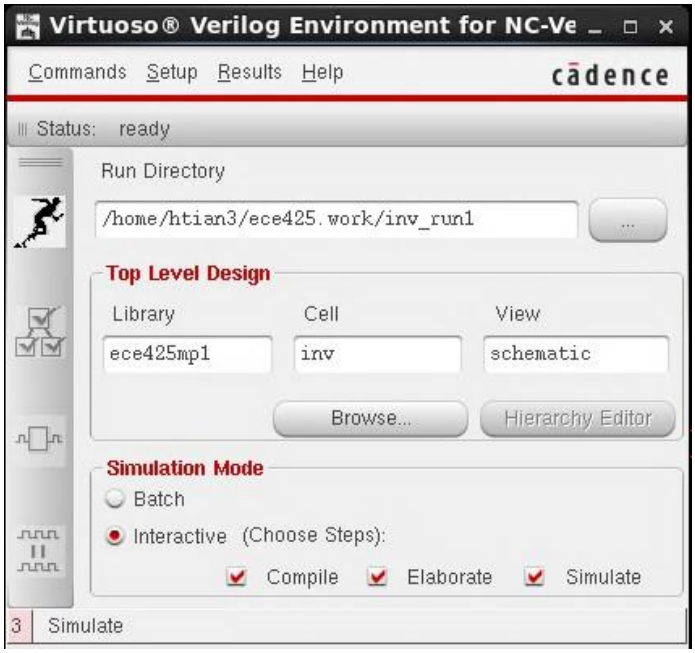
Otherwise, please ignore the above section.

First, copy the stimulus files to your work directory:

```
cp  /class/ece425/mp1/*.v  .
```

This procedure will walk you through NC-Verilog for the inverter:

1. Open the inverter schematic.
2. Select *Launch =>Plugins => Simulation => NC-Verilog*. A window as follows should pop up. In the Library, fill in *ece425mp1*, in Cell, fill in "*inv*", and fill in "*schematic*" in View.



3. Click the top left icon (running man). This initializes the design and creates a directory *inv_run1* in your work directory.
4. Click the next icon below (three check-marked squares) to generate a *netlist* for your schematic. Netlist is a Verilog file listing all your transistors and the wires (nodes, or nets) connecting them. (You can view this file using *Results => Netlist*….)
5. On terminal window, in your working directory (ece425.work), execute

```
cp  test1input.v  inv_run1/testfixture.verilog
```

2. This is to replace the default test input file to be the provided one. Note that for schematics with two inputs (e.g. nand2), the input file should be "test2inputs.v".
3. Alternatively, you can also copy via commands-Edit Test Fixture, under the "*stimulus*" field.
6. Click the third button down (rectangle with wave on sides). The button should say "Simulate" when you move mouse over it.
7. In the SimVision window that appears, click the "*Play*" button or choose "*Simulation => Run*" to simulate the design.
8. Right-click the *test* structure in the design browser and choose "*Send to Waveform Window*" to see your simulation results. You can play with the design browser and waveform window to learn the navigation features. (These will come in handy later when you have bugs to fix.)
9. Go back to the *Virtuoso Verilog Environment for NC-Verilog Integration* window and select "*Setup => Simulation Compare…*". Fill in below fields, and then click OK:

    a. Golden Database File: /class/ece425/mp1/inv_run1/shm.db/shm.db.trn
    b. Compare Database File: inv_run1/shm.db/shm.db.trn

    Golden database file is the file we have provided with the correct functionality. Compare database file is the simulation result of your schematic. You are comparing those two to find out whether the logic you have created matches golden design, which proves the correct functionality.

    By default, comparison only checks 0-100ns between your result (i.e. compare database file) and the golden design. Uncheck "*Specify Time Range of Interest*" to compare the full time span. Alternatively, manually set the end time to the end time as specified in the Verilog.

    To view a detailed report, change "*report verbosity level*" to "*detailed*".
10. Click the bottom-most button on the left (two waves with equal sign in between). The CompareScan tool should run and announce that "All variables matched". In case of the software bug, you might need to **open the file "inv_run1/compare.out" to check the result.**
11. If errors occurred, click the *plus icons* (+) to see the mismatches and identify them on the waveform. You must work backward from the waveforms to debug your schematics.
12. When you are finished, close all the NC-Verilog windows and restart from step 4.

Repeat and simulate all other cells as you did with the inverter before proceeding to layout the cells.

- Tips: You might be encountering similar errors throughout your verification process. Solutions to commonly occurred errors can be found online (forums, boards etc.), and this is a good place to start.
    - **http://www.eda.ncsu.edu/wiki/Verification**

Part 2: Layout

This will be your first real experience with layout, as the tutorial did not demand much original work. Manual layout is generally called *full-custom design*, but we still have some automated tools to help. Do not compromise the quality of your work by trying to let Cadence do everything for you. Here is a brief tutorial (the final product is not useful) on using parameterized layout cells, or *pcells*:

1. Click the *Instance* button to add a cell to the layout.
2. Browse to *NCSU_TechLib_tsmc03d => nmos*.
3. The parameters are initially set for a minimum-sized transistor. Increase the width to 1 micron by typing "1u" in the box. (This is not actually the transistor size you will use in this MP.) Observe that the number is rounded to fit manufacturing constraints and a unit (M for meters) is added.
4. Increase the *multiplier* to 2 to create a pair of parallel gates. Note that *finger* option can be used for serial connection. Difference is illustrated in the figure below.

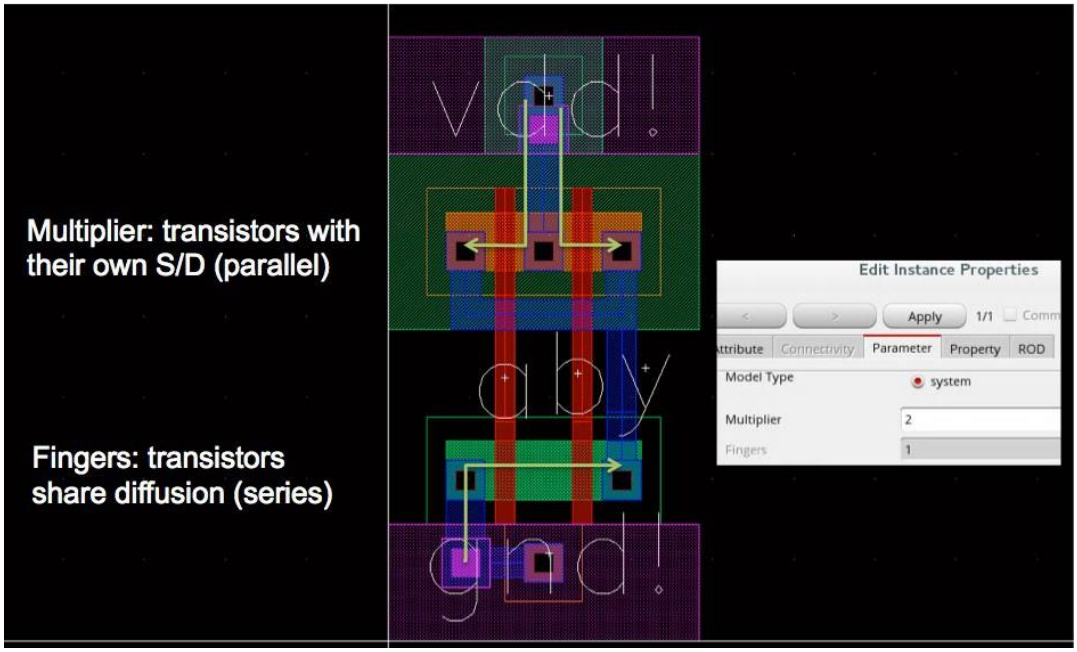


Figure. Multiplier vs. finger option in layout

5. Click on the drawing canvas to place the transistors. The pcell script is designed to make optimal use of the process rules, but it is not perfect. It used more metal 1 (dark blue) than necessary. We do not want to waste any metal 1 area, which is a precious resource for wires.
6. Click the cell and select *Edit => Hierarchy => Flatten*. Select *Flatten Pcells* and click *OK*. Now the transistors are a collection of rectangles on different layers.
7. Uncheck "s" on *nactive* in the window named "Layer" on the left side of Virtuoso to make it non-selectable as the figure below.

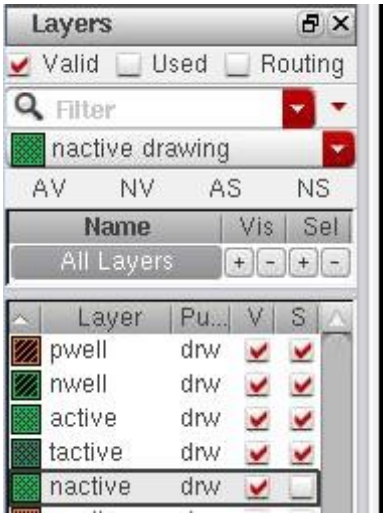


Figure. Layer window

8. Choose *Edit => Basic => Chop* (or press *shift-C*). Drag across the transistors to select the blue metal1 but not the red gates or green outer box.
9. Zoom into the cell by right-dragging across it. Press *k* and draw a ruler from the bottom of the metal1 to the bottom of the (black) contact. This is the amount of metal1 we want to leave on the top. Click the top of the contact and stretch a second ruler to equal the length of the first, so we can use it as a guide.
10. Press ESC (only once) to go back to chop mode. Click the upper left of the window, and then drag the rectangle down to touch the top of the second ruler and right to the other edge of the window. The resulted *nmos* should be looking like the following.

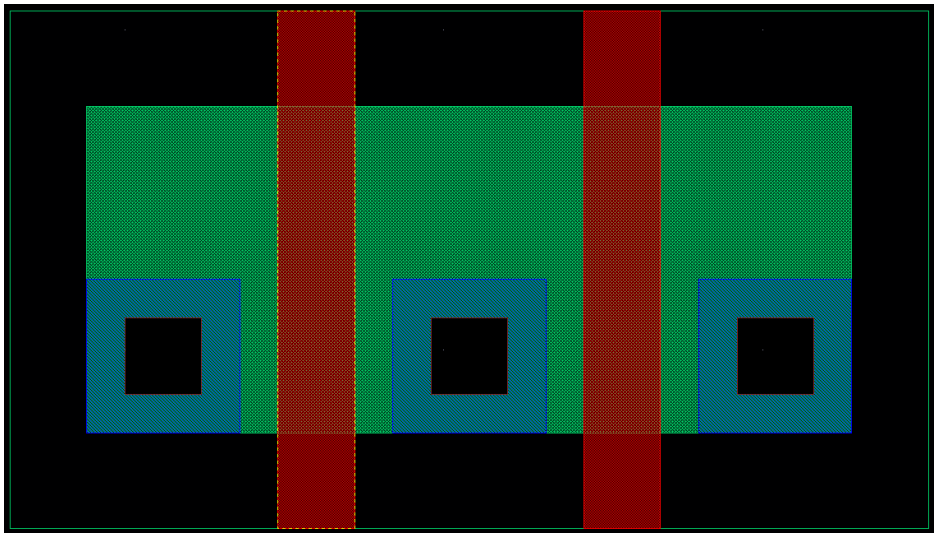


Figure. Sample NMOS with minimized metal 1

Although this still takes a while, it is far less work than starting from scratch. You should explore the pcells *nmos, pmos, m1_n, m2_m1,* etc. You can achieve better quality by being more efficient, or you can be faster by sacrificing quality. Highly recommend the former option!

Cell characteristics

Your cells will be building blocks for larger circuits, so they should fit together well and be easy to reuse. Use these guidelines to help minimize trouble later:

1. All wires and polysilicon should be minimum width. (This is the default width when you activate the path tool.) Vdd/gnd may be wider if it simplifies following the design rules.
2. Cells should have a ground (gnd!) wire along the bottom (the x axis) and a vdd! wire along the top, (example at y = 6.6 um), both in metal2. These coordinates refer to the centerline of the wire.
3. Your cells should each have one n-well. The n-well and substrate should both have body contacts (use the layout cells *ptap* and *ntap* in *NCSU_TechLib_tsmc03d*).
4. Transistor gates should be vertical. Gate poly must be 0.24u (1/2 x min poly spacing) from the power/ground wire centerlines so cells can be joined edge-to-edge (*abutted*).
5. Input pins should be on metal1 with no obstructions to the left edge of the cell. Outputs should be on metal1 with a clear path to the right edge. It will be easier to route between cells if top/bottom connections between cells are possible as well. Keeping inputs and outputs aligned horizontally simplifies the process of connecting cells.
6. Wires in metal2 should be all horizontal and used sparingly. Metal3 may not be used for now.
7. Poly jumpers are not allowed: every poly wire should connect to metal1 exactly once.

Once you have laid out a cell, you should run DRC, then extraction and LVS.

1. To run DRC, choose *Verify => DRC*, and click *OK*. To extract from the layout, choose *Verify => Extract*, and click *OK*.
2. Before running LVS, you should set the options to verify transistor sizes as well as connectivity. Choose *NCSU => Modify LVS Rules*, and check *Compare FET Parameters* and uncheck *Allow FET Series Permutation*. Then run LVS by choosing *Verify = > LVS*. Make sure the comparison is between schematic view and extracted view of the same cell before clicking *Run*. If the net-lists mismatch, you may trace the errors by clicking *Error Display*.

One common LVS error results from having a series (stack) of transistors in a different order between the layout and the schematic, for example the nmos in a NAND gate. Check for this problem by tracing from the output to the rail (power/ground) in layout and schematic, and rearranging the input pins so they match.

LVS: make sure your LVS window is Artist LVS not Diva LVS

To do this, you will need:

1. Copy simrc file into your ece425.work directory:

   cd ece425.work

   cp /class/ece425/simrc .

   alternatively, you can go to cell_design folder under mp0 directory, copy 'simrc' file in your cell-design folder, then go back to ece425.work directory, paste 'simrc' file there.
2. In your terminal, run
   export CDS_Netlisting_Mode=Analog

Part 3: Adder construction

1. Please create a cell *half_adder* from the XOR gate and the NAND gate in your library.
2. Then, create a cell *full_adder* from two *half_adders* and an additional NAND gate.
3. Finally, make an *8-bit ripple carry adder* from eight *full adders*.

The process is similar to creating the cell library, except you will use existing cells rather than transistors. Do **not** flatten cells from your library in the layout of higher-level cells, to fix bugs or to optimize. It is

imperative that you adjust the layouts of your basic cells, and use the half adder and full adder layouts only to connect the cells at the next lower level.

You should add schematic symbols for your NAND and XOR cells. It is simple and honest to borrow existing ones:

1. In the library manager, select the cell schematic and select *File => New => Cell View….*
2. Choose *schematicSymbol* as the cell type.
3. In the empty canvas, choose *Create => Import Symbol*. Select *NCSU_Digital_Parts* for the library and find an appropriate match to your cell. Click on your canvas to add the graphic.
4. Remove everything except the green symbol, the red bounding box, the red pins, and their labels. Rename the pins to a, b, y. (Be sure to edit the properties of the pins, not their labels.)
5. Choose *Edit => Origin* and click the top input pin. (This is where the cursor will appear in the preview outline when placing instances of the cell.)

An illustration of a full adder appears on the figure below. Pin names and locations are only suggestions. Remember that you also need a half adder cell, which works alone and is placed twice within the full adder layout. The half adder contains an XOR and a NAND gates and produces an inverted carry output. To share the *gnd* line between the two rows in the layout, you should try the *upside down* button when adding the bottom row.

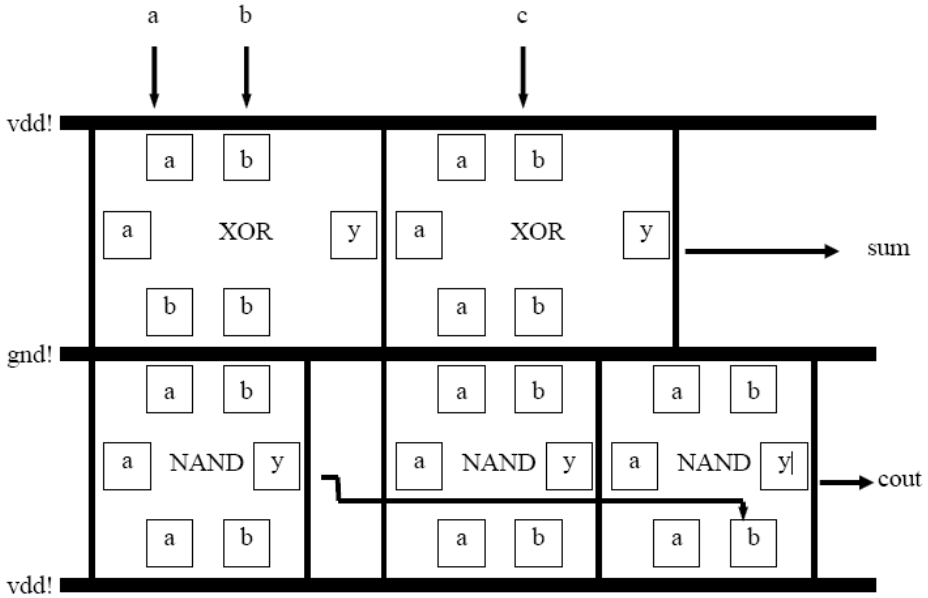


Figure. Full adder layout illustration

Possible floorplan. Try to let the pins touch as many edges as possible on the metal1 layer, to reduce dependence on metal2 (and ultimately leave more free space for other things on metal2).

Inputs			Outputs	
c	a	b	sum	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(b) Truth Table

Figure. Full Adder truth table

8-bit adder

When creating the 8-bit adder schematic, be sure to create pins a<7:0>, b<7:0>, and sum<7:0> (bits in order, not <0:7>), or Verilog will get confused. You will need the stimulus file *test17inputs.v* and golden database */class/ece425/mp1/adder_8bit_run1/shm.db/shm.db.trn*.

To layout the *8BitAdder*, stack 8 *full_adders* together (see Figure below). Since this is a "top level" layout, use metal3 to connect the inputs and outputs on the top and bottom edges as shown. (You only need to connect one, and copy the wires to the remaining seven.) Also use metal 3 to connect the top and bottom *vdd* lines. NOTE: Be sure to turn off *Join Nets With Same Name* in the extractor, since now it's cheating to assume wires are connected simply because they have the same label!

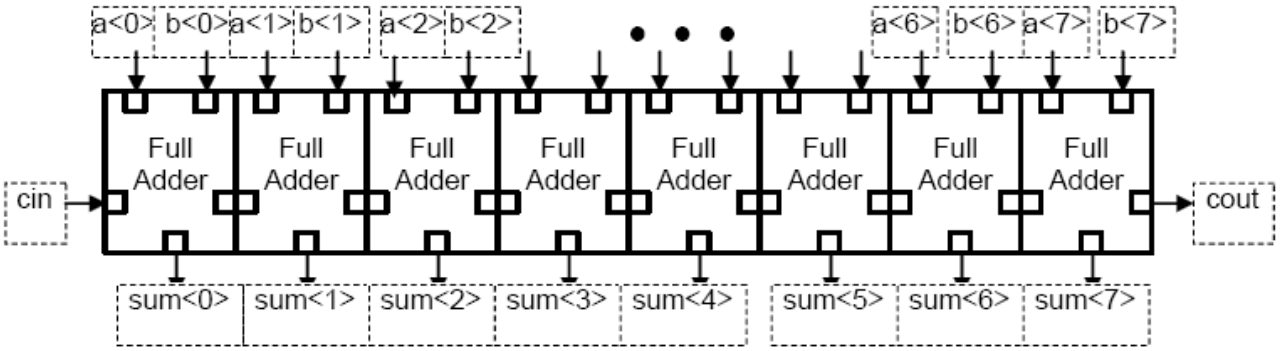


Figure. 8-Bit Adder top-level layout

MP1 Report (10 points):

Single PDF file (mp1_*netid*.pdf) containing following items in order

1. **Basic cells**
    a. **xor2:** schematic and layout
    b. **aoi22:** schematic and layout
2. **Full adder**
    a. schematic
    b. simulation waveform
    c. top-level of the layout hierarchy
3. **8-bit adder**
    a. schematic
    b. top-level layout
    c. LVS output report (si.out)
    d. Simulation comparison output report (compare.out).
    (Save the comparison report as a text file, then combine to PDF file.)
4. Please also submit all your design files in a zipped folder (such that anyone can evaluate your work in cadence).

<Submission Settings>

- For the adder layouts, only show the wires connecting the lower-level cells at the top level of the layout hierarchy. Click on *File => Print*, which will open the **Submit Plot** window. In the bottom right, click on *Plot Options* and set the levels from 0 to 0.
- To show the cell names instead of instance names (like I5) at level 1 of the hierarchy, set **Show Name Of** to *Master*.
- Include pin names of the top-level hierarchy. Make sure to turn off instance pin names.
- Also, be sure to turn off the header and turn on "fit to page" in the printing options.

<Printing instructions>

To convert output report file (text file) to PDF file, you can run following commands:

```
enscript -p si_out.ps si.out
ps2pdf si_out.ps si_out.pdf
```

Make sure to print to files and use the command line from MP0 to create single PDF document to submit the report.

<Optional guidelines>

- If your 8-bit adder doesn't work, include a comparison report for your full adder.
- If you used an alternative floorplan to those given in the figures, document it using figures similar to ours.